Introducción a la Computación

11.ª edición

J. Glenn Brookshear

ALWAYS LEARNING

PEARSON

Introducción a la computación

Introducción a la computación

Undécima edición

J. Glenn Brookshear

Colaboración de

David T. Smith

Indiana University of Pennsylvania

Dennis Brylow

Marquette University

Traducción

Vuelapluma

Revisión técnica

Gregorio Fernández Fernández

Catedrático de Ingeniería de Sistemas Telemáticos

Universidad Politécnica de Madrid

Alberto Prieto Espinosa

Catedrático de Arquitectura y Tecnología de Computadores

Universidad de Granada

Lourdes Tajes Martínez

Profesora Titular de Ciencias de la Computación e Inteligencia Artificial

Universidad de Oviedo

PEARSON

Datos de catalogación bibliográfica

INTRODUCCIÓN A LA COMPUTACIÓN, 11.ª edición
J. Glenn Brookshear

PEARSON EDUCACIÓN, S. A., Madrid, 2012

ISBN eBook: 9788478291380

Materia: 004. Informática

Formato: 195 X 250 mm. Páginas: 720

ISBN: 9788478291397
Depósito Legal:

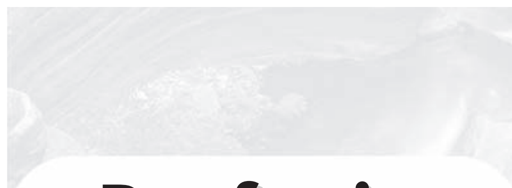
Equipo de edición:
Editor: Miguel Martín-Romo
Técnico editorial: Esther Martín

Equipo de diseño:
Diseñador: Elena Jaramillo
Técnico de diseño: Irene Medina

Equipo de producción:
Directora de producción: Marta Illescas
Jefe de producción: José A. Clares

Diseño de cubierta: Copibook, S. L.
Composición: Vuelapluma, S.L.U.
Impresión:

IMPRESO EN ESPAÑA - PRINTED IN SPAIN



Prefacio

Este libro es una introducción a las Ciencias de la computación. En él se explora la amplia variedad de temas de este campo, aunque con la suficiente profundidad como para que el lector pueda llegar a comprender los temas abordados.

A quién está dirigido el libro

He escrito este texto para estudiantes de Ciencias de la computación, así como para estudiantes de otras disciplinas. Por lo que se refiere a los estudiantes de Ciencias de la computación, la mayoría de ellos comienzan sus estudios con la ilusión de que esta disciplina tiene que ver fundamentalmente con la programación, las páginas web y la compartición de archivos a través de Internet, ya que eso es básicamente lo que ellos han visto hasta el momento. Sin embargo, el campo de las Ciencias de la computación va mucho más allá. Por ello, los estudiantes que comienzan en este campo necesitan conocer la amplitud de la materia en la que pretenden titularse. Tratar de comunicar esa amplitud es, precisamente, el objetivo de este libro. En él se proporciona a los estudiantes una panorámica de las Ciencias de la computación, una base que les permita apreciar la importancia y las interrelaciones de las futuras asignaturas que vayan cursando. Este enfoque panorámico es, de hecho, el modelo utilizado en los cursos de introducción a las Ciencias naturales.

Asimismo, este amplio repaso es lo que necesitan los estudiantes de otras disciplinas que tengan que relacionarse con la sociedad técnica en la que viven. Un curso sobre Ciencias de la computación para este tipo de audiencia debería proporcionar una comprensión práctica y realista del campo de estudio completo, en lugar de simplemente una introducción a la utilización de Internet o una formación en el uso de algunos paquetes software populares. Por supuesto, existe también un lugar apropiado para este tipo de formación, pero este texto trata acerca de la educación.

Por ello, a la hora de escribir el libro, uno de los objetivos principales era que fuese accesible para los estudiantes no técnicos. El resultado es que las ediciones anteriores se han empleado con éxito en cursos para estudiantes de un amplio rango de disciplinas y de niveles educativos, que van desde el bachillerato a los cursos de máster. Esta undécima edición está diseñada para continuar con esa tradición.

Novedades en la undécima edición

El objetivo subyacente durante el desarrollo de esta undécima edición ha sido actualizar el texto para incluir los dispositivos móviles de mano, en particular los teléfonos inteligentes. Así, el lector comprobará que el texto se ha modificado, y en ciertos aspectos ampliado, con el fin de presentar la relación entre la

materia que se está explicando y la tecnología de teléfonos inteligentes. Entre los temas específicos se incluyen:

- Hardware de los teléfonos inteligentes.
- Diferencia entre las redes 3G y 4G.
- Sistemas operativos para teléfonos inteligentes.
- Desarrollo software para teléfonos inteligentes.
- La interfaz hombre/máquina en los teléfonos inteligentes.

Estas ampliaciones son especialmete perceptibles en los Capítulos 3 (Sistemas operativos) y 4 (Redes e Internet), aunque también se pueden observar en los Capítulos 6 (Lenguajes de programación) y 7 (Ingeniería del software).

Otros cambios prominentes en esta edición incluyen la actualización de los siguientes temas:

- Propiedad y responsabilidad legal en el software: el material del Capítulo 7 (Ingeniería del software) relativo a este tema ha sido reescrito y actualizado.
- Entrenamiento de redes neuronales artificiales: este material, incluido en el Capítulo 11 (Inteligencia artificial) se ha modernizado.

Por último, el lector comprobará que todo el material incluido en el texto ha sido actualizado para reflejar la situación de la tecnología actual. Esto es especialmente acusado en el Capítulo 0 (Introducción), en el Capítulo 1 (Almacenamiento de datos) y en el Capítulo 2 (Manipulación de datos).

Organización

El texto sigue una disposición de abajo-arriba de materias que va progresando de lo más concreto a lo más abstracto: un orden que permite hacer una presentación pedagógicamente adecuada, en la que cada tema conduce al siguiente. Se comienza con los fundamentos de la codificación de la información, del almacenamiento de datos y de la arquitectura de computadoras (Capítulos 1 y 2); después se sigue con el estudio de los sistemas operativos (Capítulo 3) y de las redes de computadoras (Capítulo 4); a continuación, se investiga el tema de los algoritmos, los lenguajes de programación y el desarrollo del software (Capítulos 5 a 7); después se exploran las técnicas para mejorar la accesibilidad a la información (Capítulos 8 y 9); se analizan algunas de las principales aplicaciones de las tecnologías de computadoras en el campo de los gráficos (Capítulo 10) y se aborda el estudio del campo de la inteligencia artificial (Capítulo 11). El libro termina con una introducción de la teoría abstracta de la computación (Capítulo 12).

Aunque el texto sigue esta progresión natural, los capítulos y secciones son sorprendentemente independientes y suelen poder leerse como unidades aisladas o reordenarse con el fin de componer secuencias alternativas de estudio. De hecho, el libro se utiliza a menudo como texto en cursos diversos en los que el material se cubre en diferentes órdenes. Una de esas alternativas comienza con el material de los Capítulos 5 y 6 (Algoritmos y Lenguajes de programación) y vuelve, según sea necesario, a los capítulos anteriores. Por contraste, sé de un curso que comienza con el material sobre computabilidad del Capítulo 12. En otros casos, el texto se ha empleado como introducción a los cursos de

máster, en los que sirve como base de partida para posteriores proyectos en diferentes áreas. Los cursos para audiencias con una orientación menos técnica pueden concentrarse en los Capítulos 4 (Redes e Internet), 9 (Sistemas de bases de datos), 10 (Gráficos por computadora) y 11 (Inteligencia artificial).

En la página inicial de cada capítulo se han utilizado asteriscos para marcar algunas secciones como opcionales. Se trata de secciones que cubren temas de interés más específico o que quizá exploran temas tradicionales con una mayor profundidad. Mi intención es simplemente proporcionar sugerencias para rutas de estudio alternativas del texto. Por supuesto, existen otras posibilidades de resumir el curso. En particular, si lo que está buscando es una lectura rápida del libro, le sugiero la siguiente secuencia:

Sección	**Tema**
1.1–1.4	Fundamentos del almacenamiento y la codificación de datos
2.1–2.3	Arquitectura de las máquinas y lenguaje máquina
3.1–3.3	Sistemas operativos
4.1–4.3	Redes e Internet
5.1–5.4	Algoritmos y diseño de algoritmos
6.1–6.4	Lenguajes de programación
7.1–7.2	Ingeniería del software
8.1–8.3	Abstracciones de datos
9.1–9.2	Sistemas de bases de datos
10.1–10.2	Gráficos por computadora
11.1–11.3	Inteligencia artificial
12.1–12.2	Teoría de la computación

Existen varias ideas fundamentales que se entremezclan a lo largo del texto. Una de ellas es que las Ciencias de la computación son dinámicas. El texto presenta una y otra vez los temas desde una perspectiva histórica, explica cuál es el estado actual de las cosas e indica las direcciones en las que progresa la investigación actual. Otra de las ideas fundamentales es el papel de la abstracción y de la forma en la que se utilizan herramientas abstractas para controlar la complejidad. Este tema se presenta en el Capítulo 0 y luego se retoma en el contexto de la arquitectura de los sistemas operativos, de las redes, del desarrollo de algoritmos, del diseño de lenguajes de programación, de la ingeniería del software, de la organización de los datos y de los gráficos por computadora.

A los profesores

En este texto se incluye más material del que normalmente puede cubrirse en un único semestre, así que no dude en saltarse temas que no se adapten a los objetivos de su curso, o en reordenar los capítulos como crea oportuno. Verá que aunque el texto sigue una trama muy definida, los temas están tratados de forma bastante independiente, lo que permite seleccionarlos de la manera deseada. El libro está diseñado para ser utilizado como recurso para una asignatura, no como definición de esa asignatura. Le sugiero que anime a los estudiantes a leer aquel material que no haya sido incluido explícitamente dentro del curso. Creo que subestimaríamos a los estudiantes si supusiéramos que es

necesario explicarlo todo en el aula. Lo que los profesores debemos hacer es ayudarles a aprender por su cuenta.

Me siento obligado a dedicar unas pocas palabras a la organización del texto, que es tipo abajo-arriba, es decir, de lo concreto a lo abstracto. Creo que los profesores asumimos demasiado a menudo que los estudiantes van a apreciar nuestra perspectiva de una materia, la cual es una perspectiva que hemos desarrollado a lo largo de muchos años de trabajar en un cierto campo. Como profesores, creo que haríamos mejor presentando el material desde la perspectiva de los estudiantes. Esta es la razón por la que el texto comienza con la representación/almacenamiento de datos, la arquitectura de las máquinas, los sistemas operativos y las redes. Estos son temas que los estudiantes pueden comprender fácilmente, ya que lo más probable es que ya hayan oído términos tales como JPEG y MP3, hayan grabado datos en discos CD y DVD, adquirido componentes de computadora, interactuado con un sistema operativo y utilizado Internet. Iniciando el curso con estos temas, los estudiantes descubren las respuestas a muchos de los "porqués" que han estado planteándose durante años y aprenden a ver el curso como algo práctico más que teórico. Con este punto de partida, resulta natural continuar con las cuestiones más abstractas, relativas a los algoritmos, las estructuras algorítmicas, los lenguajes de programación, las metodologías del desarrollo de software, la computabilidad y la complejidad, y son precisamente estas cuestiones las que los que trabajamos en este campo consideramos como los temas principales de esta disciplina. Como he dicho antes, los temas se presentan en el libro de forma tal que el profesor no está obligado a seguir esta secuencia abajo-arriba, pero yo me atrevería a animarle a que al menos lo intentara.

Todos somos conscientes de que los estudiantes aprenden mucho más de lo que nosotros les enseñamos directamente, y las lecciones que los estudiantes aprenden de manera implícita suelen absorberse mejor que aquellas que se estudian explícitamente. Esto es importante a la hora de "enseñar" técnicas de resolución de problemas. Los estudiantes no aprenden a resolver problemas estudiando metodologías de resolución de problemas; como aprenden a resolver problemas es resolviéndolos, y no me refiero únicamente a los típicos "problemas elegidos" que se incluyen en los libros de texto. Así que en este libro he incluido numerosos problemas, algunos de los cuales son intencionadamente vagos, lo que quiere decir que no necesariamente tiene que existir una única solución correcta o una única respuesta correcta. Animo a los profesores a utilizar estos problemas y a ampliarlos si es preciso.

Otro tema relacionado con la cuestión del "aprendizaje implícito" es el de la profesionalidad, la ética y la responsabilidad social. No creo que este material tenga que ser presentado como un tema aislado que simplemente se toca de pasada en el curso. En lugar de ello, debe ser una parte integrante de todos los temas que deberá plantearse cuando sea relevante. Este es precisamente el enfoque seguido en el texto. Podrá comprobar que las Secciones 3.5, 4.5, 7.8, 9.7 y 11.7 presentan temas tales como la seguridad, la intimidad, la responsabilidad legal y la conciencia social en el contexto de los sistemas operativos, las redes, los sistemas de bases de datos, la ingeniería del software y la inteligencia artificial. Además, la Sección 0.6 presenta este tema resumiendo algunas de las teorías más prominentes que tratan de otorgar una base filosóficamente sólida a la cuestión de una toma de decisiones ética. También verá que cada capítulo incluye un conjunto de preguntas bajo el epígrafe ***Cuestiones sociales***, que

intentan animar a los estudiantes a pensar acerca de las relaciones entre el material presentado en el texto y la sociedad en la que viven.

Gracias por tomar en consideración mi libro para su curso. Decida o no que el libro es adecuado para su situación concreta, espero que aprecie que el libro trata de contribuir a mejorar la educación en el campo de las Ciencias de la computación.

Características pedagógicas

Este texto es el resultado de muchos años de trabajo en el campo de la enseñanza. Como consecuencia, abundan en él las ayudas pedagógicas. De especial importancia es la abundancia de problemas que intentan incrementar la participación del estudiante, más de 1000 en esta undécima edición. Están clasificados como Cuestiones/ejercicios, Problemas de repaso y Cuestiones sociales. Las Cuestiones/ejercicios aparecen la final de cada sección, excepto en el capítulo de introducción. En ellas se repasa el material que se acaba de presentar, se amplían las correspondientes explicaciones o se apunta a temas relacionados que se tratarán posteriormente. Las respuestas a estas cuestiones están incluidas en el Apéndice F.

Los Problemas de repaso están incluidos al final de cada capítulo (excepto en el capítulo de introducción). Están diseñados para servir como problemas que el estudiante debe trabajar en casa, en el sentido de que cubren el material presentado a lo largo de todo el capítulo y no se responden dentro del texto.

También al final de cada capítulo se han incluido una serie de cuestiones pertenecientes a la categoría de Cuestiones sociales. Están diseñadas para que los estudiantes piensen y discutan entre ellos. Muchas de esas cuestiones pueden utilizarse como base para trabajos individuales o en grupo que culminen en cortas presentaciones orales o escritas.

Cada capítulo termina con una lista de Lecturas adicionales, que contiene referencias a otros materiales relacionados con el tema del capítulo. Los sitios web identificados en este prefacio, en el texto y en los recuadros son también lugares adecuados en los que buscar material relacionado.

Recursos suplementarios

En el sitio web de acompañamiento de este libro (www.pearsonhighered.com/brookshear) podrá encontrar diversos tipos de materiales suplementarios para este texto. Los siguientes materiales están accesibles para todos los lectores:

- Actividades para cada capítulo que amplían los temas del texto y proporcionan la oportunidad de explorar temas relacionados.
- Exámenes de auto-evaluación para cada capítulo que ayudan a los lectores a pensar más en profundidad en el material presentado en el texto.
- Manuales para enseñar los fundamentos de Java y C++ en una secuencia pedagógica compatible con el texto.

Además, los siguientes suplementos están disponibles para los profesores en el Centro de recursos del profesor (***Instructor Resource Center***) de Pearson Education. Visite www.pearsonhighered.com o contacte con su representante de ventas de Pearson para obtener información sobre cómo acceder a ese material:

- Guía del profesor (***Instructor's Guide***) con respuestas a los Problemas de repaso.
- Presentaciones PowerPoint.
- Banco de pruebas (Test bank).

También puede visitar mi sitio web personal en www.mscs.mu.edu/~glennb. No es un sitio muy formal (y está sujeto a mis caprichos y mi sentido del humor, pero suelo mantener allí alguna información que puede que le resulte útil. En particular, podrá encontrar una página de erratas donde se indican las correcciones a los errores del texto (versión del libro en inglés) que me han ido haciendo llegar.

A los estudiantes

Yo soy bastante poco conformista (algunos de mis amigos dirían que soy ***muy*** inconformista), por lo que cuando comencé a escribir este libro no siempre seguí los consejos que me dieron. En particular, muchos argumentaban que determinado material era demasiado avanzado para los estudiantes principiantes. Pero yo creo que si un tema es relevante, entonces lo seguirá siendo incluso si la comunidad académica considera que es un "tema avanzado". Creo que el estudiante se merece un texto que presente una imagen completa de las Ciencias de la computación, no una versión edulcorada que contenga presentaciones artificialmente simplificadas de únicamente aquellos temas que se hayan considerado apropiados para los estudiantes principiantes. Por tanto, no he evitado ningún tema. En lugar de ello, lo que he intentado es dar mejores explicaciones. He tratado de proporcionar un análisis con la suficiente profundidad como para que el estudiante disponga de una imagen adecuadamente precisa de lo que son las Ciencias de la computación. Al igual que sucede con las especias en una receta de cocina, el estudiante puede saltarse algunos de los temas contenidos en las siguientes páginas, pero ahí están esos temas para que los pruebe si lo desea, y yo le animo a que lo haga.

También habría que decir que en cualquier curso relacionado con la tecnología, los detalles que se aprenden hoy pueden no ser los detalles que nos harán falta mañana. Estamos hablando de un campo muy dinámico, y esa es precisamente, una de las razones por las que es tan interesante. Este libro le proporcionará una imagen actual de la materia, así como una perspectiva histórica. Con esa formación estará preparado para ir progresando a medida que la tecnología lo haga. Le animo a que comience con ese progreso ya mismo, explorando los temas más allá de lo tratado en el libro. Aprenda a aprender.

Gracias por la confianza que me ha otorgado al decidir leer mi libro. Como autor, tengo la obligación de escribir un libro que merezca el tiempo que usted le dedique. Espero haber cumplido como mi obligación y que usted pueda percibirlo al leerlo.

Agradecimientos

En primer lugar deseo dar las gracias a aquellos de ustedes que han dado su apoyo a este libro leyéndolo y utilizándolo en sus ediciones anteriores. Es para mi todo un honor.

David T. Smith (Universidad de Indiana en Pensilvania) y Dennis Brylow (Universidad de Marquette) han desempeñado un papel significativo en la producción de esta undécima edición. David se concentró en los Capítulos 0, 1, 2, 7 y 11; y Dennis en los Capítulos 3, 4, 6 y 10. Sin el duro trabajo que han realizado, esta nueva edición no existiría. Les doy las gracias sinceramente.

Como se menciona en el prefacio a la décima edición, estoy en deuda con Ed Angel, John Carpinelli, Chris Fox, Jim Kurose, Gary Nutt, Greg Riccardi y Patrick Henry Winston por su ayuda en el desarrollo de esa edición. El resultado de sus esfuerzos sigue siendo visible en esta undécima edición.

Otras personas que han contribuido a esta o anteriores ediciones son J. M. Adams, C. M. Allen, D. C. S. Allison, R. Ashmore, B. Auernheimer, P. Bankston, M. Barnard, P. Bender, K. Bowyer, P. W. Brashear, C. M. Brown, H. M Brown, B. Calloni, M. Clancy, R. T. Close, D. H. Cooley, L. D. Cornell, M. J. Crowley, F. Deek, M. Dickerson, M. J. Duncan, S. Ezekiel, S. Fox, N. E. Gibbs, J. D. Harris, D. Hascom, L. Heath, P. B. Henderson, L. Hunt, M. Hutchenreuther, L. A. Jehn, K. K. Kolberg, K. Korb, G. Krenz, J. Liu, T. J. Long, C. May, J. J. McConnell, W. McCown, S. J. Merrill, K. Messersmith, J. C. Moyer, M. Murphy, J. P. Myers, Jr., D. S. Noonan, W. W. Oblitey, S. Olariu, G. Rice, N. Rickert, C. Riedesel, J. B. Rogers, G. Saito, W. Savitch, R. Schlafly, J. C. Schlimmer, S. Sells, G. Sheppard, Z. Shen, J. C. Simms, M. C. Slattery, J. Slimick, J. A. Slomka, D. Smith, J. Solderitsch, R. Steigerwald, L. Steinberg, C. A. Struble, C. L. Struble, W. J. Taffe, J. Talburt, P. Tonellato, P. Tromovitch, E. D. Winter, E. Wright, M. Ziegler y un anónimo. Mi más sincero agradecimiento a todas estas personas.

Ya he mencionado que en el sitio web de acompañamiento podrá encontrar manuales de Java y C++ que enseñan los fundamentos de estos lenguajes con un formato compatible con el texto. Estos manuales han sido escritos por Diane Christie. Gracias Diane. Gracias también a Roger Eastman, que fue el motor creativo detrás de las actividades para los distintos capítulos que podrá encontrar también en el sitio web de acompañamiento.

También quiero dar la gracias a las personas de Addison-Wesley que han contribuido a este proyecto. Son un gran equipo con el que trabajar, además de unos buenos amigos. Si está pensando en escribir un libro de texto, debería tomar en consideración que se lo publique Addison-Wesley.

Continúo estando agradecido a mi esposa Earlene y a mi hija Cheryl que han sido un gran apoyo para mí a lo largo de los años. Por supuesto, Cheryl creció y dejó nuestro hogar hace ya años, pero Earlene sigue estando ahí. Soy un hombre con suerte. En la mañana del 11 de diciembre de 1998, sobreviví a un ataque al corazón porque ella consiguió llevarme al hospital a tiempo. (Para los lectores más jóvenes, debo explicar que el sobrevivir a un ataque al corazón es algo así como que te concedan un plazo adicional para terminar uno de los trabajos del curso.)

Por último, gracias a mis padres a los que está dedicado este libro. Permítanme que termine con el siguiente mensaje de apoyo, cuyo origen permanecerá anónimo: “El libro de nuestro hijo es realmente bueno. Todo el mundo debería leerlo.”

J. G. B.

Contenido

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

CAPÍTULO

0

Introducción

En este capítulo preliminar vamos a analizar el campo de las Ciencias de la computación, a exponer una perspectiva histórica del mismo y a establecer las bases a partir de las cuales iniciaremos nuestro estudio.

Las Ciencias de la computación o Informática es la disciplina que trata de establecer una base científica para temas tales como el diseño asistido por computadora, la programación de computadoras, el procesamiento de la información, las soluciones algorítmicas de problemas y el propio proceso algorítmico. Proporciona los fundamentos para las aplicaciones informáticas actuales, así como la base para la infraestructura de computación del futuro.

Este libro proporciona una introducción exhaustiva a esta ciencia. Investigaremos un amplio rango de temas, incluyendo la mayoría de los que componen el currículum de los estudios universitarios típicos de Ciencias de la computación. Queremos abarcar el ámbito completo de este campo, así como la dinámica del mismo. Por ello, además de los propios temas, trataremos de analizar su desarrollo histórico, el estado actual de las investigaciones y las perspectivas de futuro. Nuestro objetivo es conseguir una comprensión de las Ciencias de la computación desde el punto de vista funcional; una comprensión que permita al lector continuar con estudios más especializados en esta área, y que también permita a aquellos que trabajan en otros campos sobrevivir en una sociedad cada vez más tecnificada.

0.1 El papel de los algoritmos

Comenzaremos con el concepto más fundamental de las Ciencias de la computación: el concepto de algoritmo. Informalmente, un **algoritmo** es un conjunto de pasos que define cómo hay que realizar una tarea. (Seremos más precisos a este respecto en el Capítulo 5.) Por ejemplo, existen algoritmos para cocinar (recetas), para encontrar el camino en una ciudad desconocida (direcciones), para hacer funcionar una lavadora (instrucciones que normalmente pueden encontrarse en el manual), para tocar música (expresadas mediante partituras) y para realizar trucos de magia (Figura 0.1).

Para que una máquina como una computadora pueda llevar a cabo una tarea, es preciso diseñar y representar un algoritmo de realización de dicha tarea y en una forma que sea compatible con la máquina. A la representación de un algoritmo se la denomina **programa**. Por comodidad de los seres humanos, los programas informáticos suelen imprimirse en papel o visualizarse en las pantallas de las computadoras. Sin embargo, para comodidad de las máquinas, los programas se codifican de una manera compatible con la tecnología a partir de la cual esté construida la máquina. El proceso de desarrollo de un programa, de codificarlo en un formato compatible con la máquina y de introducirlo en una máquina se denomina **programación**. Los programas y los algoritmos que representan se denominan colectivamente **software**, por contraste con la propia máquina que se conoce con el nombre de **hardware**.

El estudio de los algoritmos comenzó siendo un tema del campo de las matemáticas. De hecho, la búsqueda de algoritmos fue una actividad de gran importancia para los matemáticos mucho antes del desarrollo de las computadoras actuales. El objetivo era determinar un único conjunto de instrucciones que describiera cómo resolver todos los problemas de un tipo concreto. Uno de los ejemplos mejor conocidos de estas investigaciones pioneras es el algoritmo de división para el cálculo del cociente de dos números de varios dígitos. Otro ejemplo es el algoritmo de Euclides descubierto por este matemático de la anti-

Figura 0.1 Algoritmo para un truco de magia.

Efecto: el mago coloca boca abajo sobre una mesa algunas cartas extraídas de un mazo de cartas normal y las baraja suficientemente, distribuyéndolas después sobre la mesa. Después, a medida que el público solicita cartas rojas o negras, el mago da la vuelta a cartas del color solicitado.

El truco:

Paso 1. De un mazo de cartas normal, seleccione diez cartas rojas y diez negras. Coloque estas cartas boca arriba sobre la mesa en dos montones, de acuerdo con su color.

Paso 2. Anuncie que ha seleccionado algunas cartas rojas y algunas negras.

Paso 3. Tome las cartas rojas. Con el pretexto de alinear el mazo formado por esas cartas, manténgalas boca abajo en su mano izquierda y, con el pulgar y el índice de su mano derecha tire de cada extremo del mazo, de modo que cada carta quede ligeramente curvada *hacia atrás*. Luego, coloque el mazo de cartas rojas boca abajo sobre la mesa al mismo tiempo que dice: "En este mazo tenemos las cartas rojas".

Paso 4. Tome las cartas negras. De forma similar a como ha hecho en el Paso 3, proporcione a estas cartas una ligera curvatura *hacia adelante*. Luego devuelva estas cartas a la mesa colocando el mazo boca abajo al mismo tiempo que dice: "Y en este otro mazo están las cartas negras".

Paso 5. Inmediatamente después de colocar las cartas negras sobre la mesa, utilice ambas manos para mezclar las cartas negras y rojas (que todavía están boca abajo) y distribuirlas sobre la mesa. Explique que está mezclando convenientemente las cartas.

Paso 6. Mientras que haya cartas boca abajo sobre la mesa, ejecute repetidamente los siguientes pasos:

6.1. Pida a alguien del público que solicite una carta roja o negra.

6.2. Si el color solicitado es rojo y hay alguna carta boca abajo con apariencia cóncava, dé la vuelta a esa carta al mismo tiempo que dice "He aquí una carta roja".

6.3. Si el color solicitado es negro y hay alguna carta boca abajo con apariencia convexa, dé la vuelta a dicha carta al mismo tiempo que dice "He aquí una carta negra".

6.4. En caso contrario, anuncie que ya no hay más cartas del color solicitado y dé la vuelta a las cartas restantes para demostrar su afirmación.

gua Grecia que permite determinar el máximo común divisor de dos números enteros positivos (Figura 0.2).

Una vez que se ha encontrado un algoritmo para llevar a cabo una determinada tarea, la realización de esta ya no requiere comprender los principios en los que el algoritmo está basado. En lugar de ello, la realización de la tarea se reduce al proceso de seguir simplemente las instrucciones proporcionadas. (Podemos emplear el algoritmo de división para calcular un cociente o el algoritmo de Euclides para hallar el máximo común divisor sin necesidad de entender por qué funciona el algoritmo.) En cierto sentido, la inteligencia requerida para resolver ese problema está codificada dentro del algoritmo.

Es esta capacidad de capturar y transmitir inteligencia (o al menos un comportamiento inteligente) por medio de algoritmos lo que nos permite construir máquinas que lleven a cabo tareas de utilidad. En consecuencia, el nivel de

Figura 0.2 El algoritmo de Euclides para calcular el máximo común divisor de dos números enteros positivos.

Descripción: este algoritmo presupone que la entrada está formada por dos enteros positivos y calcula el máximo común divisor de dichos valores.

Procedimiento:

Paso 1. Asigne a M y N el valor del mayor y el menor de los dos números proporcionados como entrada, respectivamente.

Paso 2. Divida M entre N y llame R al resto.

Paso 3. Si R es distinto de 0, entonces asigne a M el valor de N, asigne a N el valor de R y vuelva al Paso 2. En caso contrario, el máximo común divisor será el valor asignado actualmente a N.

inteligencia mostrado por las máquinas está limitado por la inteligencia que podamos transmitir mediante algoritmos. Solo podemos construir una máquina para llevar a cabo una tarea si existe un algoritmo que permita realizar esa tarea. A su vez, si no existe ningún algoritmo para resolver un problema, entonces la solución de ese problema cae fuera de la capacidad de las máquinas disponibles.

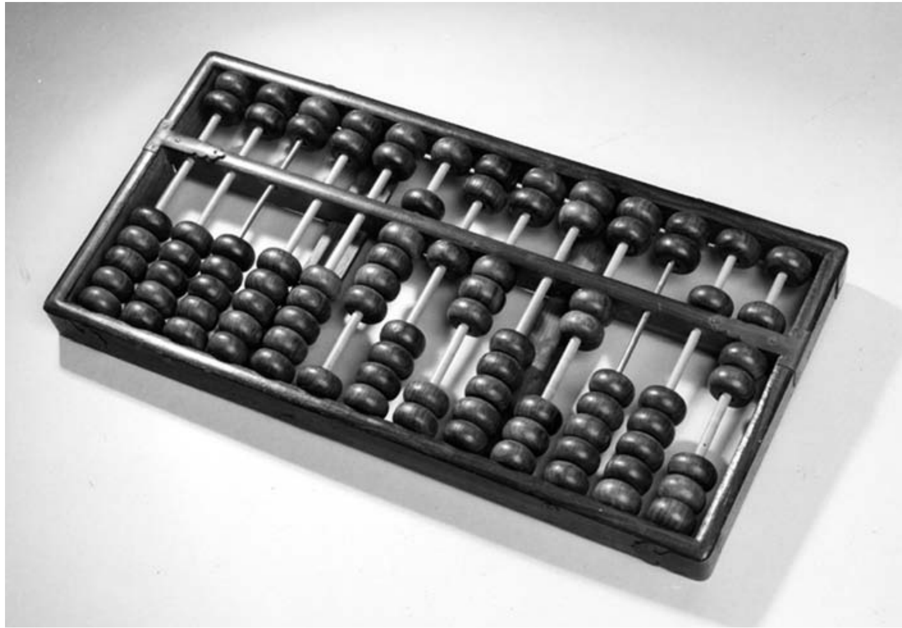
La identificación de las limitaciones de las capacidades algorítmicas terminó convirtiéndose en uno de los temas de las matemáticas en la década de 1930 con la publicación del teorema de incompletitud de Kurt Gödel. Este teorema afirma, en esencia, que en cualquier teoría matemática que abarque nuestro sistema aritmético tradicional, hay enunciados cuya verdad o falsedad no puede establecerse por medios algorítmicos. Por decirlo en pocas palabras, cualquier estudio completo de nuestro sistema aritmético, cae fuera de las capacidades de las actividades algorítmicas.

El enunciado de este hecho removió los cimientos de las matemáticas y el estudio de las capacidades algorítmicas que se inició a partir de ahí fue el comienzo del campo que hoy día conocemos como Ciencias de la computación. De hecho, es el estudio de los algoritmos lo que forma la base fundamental de estas ciencias.

0.2 La historia de la computación

Las computadoras actuales tienen una genealogía muy extensa. Uno de los primeros dispositivos de computación fue el ábaco. La historia nos dice que sus raíces se hunden, muy probablemente, en la antigua China y fue utilizado por las antiguas civilizaciones griega y romana. Dicha máquina es muy simple, estando compuesta por una serie de cuentas ensartadas en unas varillas que a su vez se montan sobre un marco rectangular (Figura 0.3). Al mover las cuentas hacia adelante y hacia atrás en las varillas, sus posiciones representan los valores almacenados. Es gracias a las posiciones de las cuentas que esta “computadora” representa y almacena los datos. Para el control de la ejecución de un algoritmo, esta máquina depende del operador humano. Por tanto, el ábaco es, por sí solo, un sistema de almacenamiento de datos; se precisa la intervención de una persona para poder disponer de una máquina de computación completa.

Figura 0.3 Un ábaco (fotografía de Wayne Chandler).



En el periodo posterior a la Edad Media y anterior a la Edad Moderna, se sentaron las bases para la búsqueda de máquinas de computación más sofisticadas. Unos cuantos inventores comenzaron a experimentar con la tecnología de los engranajes. Entre ellos estaban Blaise Pascal (1623–1662) en Francia, Gottfried Wilhelm Leibniz (1646–1716) en Alemania y Charles Babbage (1792–1871) en Inglaterra. Estas máquinas representaban los datos mediante posicionamiento con engranajes, introduciéndose los datos mecánicamente por el procedimiento de establecer las posiciones iniciales de esos engranajes. En las máquinas de Pascal y Leibniz, la salida se conseguía observando las posiciones finales de los engranajes. Babbage, por su parte, concibió máquinas que imprimían los resultados de los cálculos en papel, con el fin de poder eliminar la posibilidad de que se produjeran errores de transcripción.

Por lo que respecta a la capacidad de seguir un algoritmo, podemos ver una cierta progresión en la flexibilidad de estas máquinas. La máquina de Pascal se construyó para realizar únicamente sumas. En consecuencia, la secuencia apropiada de pasos estaba integrada dentro de la propia estructura de la máquina. De forma similar, la maquina de Leibniz tenía los algoritmos firmemente integrados en su arquitectura, aunque ofrecía diversas operaciones aritméticas entre las que el operador podría seleccionar una. La máquina diferencial de Babbage (de la que solo se construyó un modelo de demostración) podía modificarse para realizar diversos cálculos, pero su máquina analítica (para cuya construcción nunca consiguió financiación) estaba diseñada para leer las instrucciones en forma de agujeros realizados en una tarjeta de cartón. Por tanto, la máquina analítica de Babbage era programable. De hecho, se considera a Augusta Ada Byron (Ada

Lovelace), que publicó un artículo en el que ilustraba cómo podría programarse la máquina analítica de Babbage para realizar diversos cálculos, como la primera programadora del mundo.

La idea de comunicar un algoritmo mediante agujeros en una tarjeta de cartón perforada no era original de Babbage. Él tomó esa idea de Joseph Jacquard (1752–1834) que, en 1801, había desarrollado un telar en el que los pasos que había que realizar en el proceso de tejido estaban determinados por una serie de patrones de agujeros realizados en grandes y gruesas tarjetas hechas de madera (o cartón). De esta forma, el algoritmo seguido por el telar podía modificarse fácilmente para producir diferentes diseños de tejidos. Otra persona que se

Máquina diferencial de Babbage

Las máquinas diseñadas por Charles Babbage fueron los verdaderos antecedentes de los modernos diseños de computadoras. Si hubiera estado disponible la tecnología para fabricar sus máquinas de forma económicamente factible y si las demandas de procesamiento de datos de la industria y del gobierno hubieran tenido una escala similar a la actual, las ideas de Babbage podrían haber conducido a una revolución informática en el siglo xix. Sin embargo, lo que sucedió es que solo se pudo construir un modelo de demostración de su máquina diferencial durante su vida. Esta máquina determinaba los valores numéricos calculando "diferencias sucesivas". Podemos tratar de entender esta técnica considerando el problema de calcular los cuadrados de los números enteros. Partimos del hecho de que el cuadrado de 0 es 0, el cuadrado de 1 es 1, el cuadrado de 2 es 4 y el cuadrado de 3 es 9. Sabiendo esto, podemos determinar el cuadrado de 4 de la forma siguiente (vea el siguiente diagrama). Primero calculamos la diferencia de los cuadrados que ya conocemos: $1^2 - 0^2 = 1$, $2^2 - 1^2 = 3$ y $3^2 - 2^2 = 5$. A continuación calculamos la diferencias de estos resultados: $3 - 1 = 2$ y $5 - 3 = 2$. Observe que estas diferencias son ambas iguales a 2. Suponiendo que dicho valor continúe siendo constante (el cálculo matemático nos permite verificarlo) concluimos que la diferencia entre el valor $(4^2 - 3^2)$ y el valor $(3^2 - 2^2)$ tiene que ser también 2. Por tanto, $(4^2 - 3^2)$ tiene que ser 2 unidades mayor que $(3^2 - 2^2)$, por lo que $4^2 - 3^2 = 7$ y por tanto $4^2 = 3^2 + 7 = 16$. Ahora que conocemos el cuadrado de 4, podemos continuar con este mismo procedimiento para calcular el cuadrado de 5 basándonos en los valores de 1^2, 2^2, 3^2 y 4^2. (Aunque una discusión más detallada acerca de las diferencias sucesivas queda fuera del alcance de este estudio, los estudiantes de cálculo habrán observado que el ejemplo anterior se basa en el hecho de que la derivada de $y = x^2$ es una línea recta con una pendiente igual a 2.)

x	x^2	Primera diferencia	Segunda diferencia
0	0	1	2
1	1	3	2
2	4	5	2
3	9	7	2
4	16		
5			

Augusta Ada Byron

Augusta Ada Byron, Condesa de Lovelace, ha sido objeto de una gran atención y de numerosos análisis en la comunidad informática. Vivió una vida un tanto trágica, que no llegó a los 37 años (1815–1852), complicada por su mala salud y por el hecho de que era una inconformista en una sociedad que limitaba el papel profesional de las mujeres. Aunque estaba interesada en un amplio rango de temas científicos, concentró sus estudios en el campo de las matemáticas. Su interés en la "Ciencia de la computación" comenzó cuando quedó fascinada por las máquinas de Charles Babbage en la demostración de un prototipo de su máquina diferencial en 1833. Su contribución a la Ciencia de la computación surge de su traducción del francés al inglés de un artículo en el que se comentaban los diseños de Babbage relativos a su máquina analítica. Babbage animó a Ada a que añadiera a dicha traducción un apéndice en el que se describieran las aplicaciones de la máquina y que contuviera ejemplos de cómo podría programarse la máquina para realizar disversas tareas. El entusiasmo de Babbage por los trabajos de Ada Byron estaba motivado, aparentemente, por la esperanza de que su publicación pudiera conseguirle el respaldo financiero necesario para la construcción de su máquina analítica (como hija de Lord Byron, Ada Byron era toda una celebridad y disponía de conexiones potencialmente importantes con personas acomodadas). Dicho respaldo financiero nunca llegó a materializarse, pero el apéndice escrito por Ada Byron se conserva y se considera que contiene los primeros ejemplos de programas de computadora. Existe una cierta discusión entre los historiadores acerca del grado en el que el propio Babbage influyó sobre los trabajos de Ada Byron. Algunos argumentan que Babbage realizó importantes contribuciones, mientras que otros afirman que fue más un obstáculo que una ayuda para Ada Byron. En cualquier caso, hoy día se reconoce a Augusta Ada Byron como la primera programadora de la historia, un estatus que fue certificado por el Departamento de Defensa de Estados Unidos cuando llamó Ada a un importante lenguaje de programación en su honor.

aprovechó de la idea de Jacquard fue Herman Hollerith (1860–1929), que aplicó el concepto de representar la información mediante agujeros en tarjetas de cartón para acelerar el proceso de tabulación de resultados en el censo de Estados Unidos de 1890. (Fue este trabajo de Hollerith el que condujo a la creación de la empresa IBM.) Dichas tarjetas terminaron siendo conocidas con el nombre de tarjetas perforadas y sobrevivieron como método popular de comunicación con las computadoras hasta bien avanzada la década de 1970. De hecho, dicha técnica sigue empleándose hoy día, como hemos podido comprobar mediante los problemas suscitados en el recuento de las votaciones para la elección de presidente de los Estados Unidos en el año 2000.

La tecnología disponible en aquella época no permitía producir las complejas máquinas basadas en engranajes de Pascal, Leibniz y Babbage de manera económica. Pero los avances experimentados por la electrónica a principios del siglo XX permitieron eliminar dichos obstáculos. Como ejemplo del progreso efectuado en dicha época podemos citar la máquina electromecánica de George Stibitz, completada en 1940 en los Bell Laboratories, y la computadora Mark I, que se terminó de desarrollar en 1944 en la universidad de Harvard, siendo sus creadores Howard Aiken y un grupo de ingenieros de IBM (Figura 0.4). Estas máquinas hacían un uso intensivo de relés mecánicos controlados electrónicamente. En este sentido, dichas máquinas se habían quedado obsoletas casi en el mismo

Figura 0.4 La computadora Mark I. (Cortesía de los archivos de IBM. Su uso no autorizado no está permitido.)



momento en que fueron construidas, porque otros investigadores ya estaban utilizando la tecnología de los tubos de vacío para construir computadoras totalmente electrónicas. La primera de estas máquinas fue, aparentemente, la máquina de Atanasoff-Berry, construida durante el periodo de 1937 a 1941 en el Iowa State College (que ahora es la universidad estatal de Iowa) por John Atanasoff y su ayudante, Clifford Berry. Otro ejemplo es la máquina denominada Colossus, construida bajo la dirección de Tommy Flowers en Inglaterra para decodificar los mensajes alemanes durante los últimos años de la Segunda Guerra Mundial. (De hecho, aparentemente se construyeron hasta diez de estas máquinas, pero la necesidad de ocultar los secretos militares y las cuestiones de seguridad nacional impidieron que se registrara su existencia en el "árbol genealógico de las computadoras".) A estas máquinas pronto les siguieron otras más flexibles, como la computadora ENIAC (***Electronic Numerical Integrator and Calculator***, Calculador e integrador numérico electrónico) desarrollada por John Mauchly y J. Presper Eckert en la Escuela de Ingeniería Eléctrica Moore de la universidad de Pensilvania.

A partir de ahí, la historia de las computadoras ha estado estrechamente ligada a los avances tecnológicos, incluyendo la invención de los transistores (por la que obtuvieron el Premio Nobel los físicos William Shockley, John Bardeen y Walter Brattain) y el subsiguiente desarrollo de circuitos completos construidos como una unidad, denominados circuitos integrados (por los que Jack Kilby también obtuvo el Premio Nobel en Física). Con estos desarrollos, las máquinas de la década de 1940, que tenían el tamaño de una habitación, se redujeron a lo largo de las décadas siguientes hasta el tamaño de un armario. Al mismo tiempo, la potencia de procesamiento de las computadoras comenzó a duplicarse cada dos años (una tendencia que ha continuado hasta nuestros días). A medida que fue progresando el desarrollo de los circuitos integrados, muchos de los circuitos que forman parte de una computadora pasaron a estar

disponibles comercialmente en forma de circuitos integrados encapsulados en unos bloques de plástico diminutos denominados chips.

Uno de los pasos principales que condujo a la popularización de la computación fue el desarrollo de las computadoras de sobremesa. Los orígenes de estas máquinas están en los aficionados a la computación que se dedicaban a construir computadoras caseras a partir de combinaciones de chips. Fue en este mercado de "aficionados" donde Steve Jobs y Stephen Wozniak construyeron una computadora doméstica comercialmente viable y fundaron, en 1976, Apple Computer, Inc. (ahora denominada Apple Inc.) para fabricar y distribuir sus productos. Otras empresas que distribuían productos similares eran Commodore, Heathkit y Radio Shack. Aunque estos productos eran muy populares entre los aficionados a las computadoras, no fueron ampliamente aceptados al principio por la comunidad empresarial, que continuó acudiendo a la empresa IBM, de gran renombre, para satisfacer la mayor parte de sus necesidades de computación.

En 1981, IBM presentó su primera computadora de sobremesa, denominada computadora personal o PC (*Personal Computer*), y cuyo software subyacente había sido desarrollado por una empresa de reciente creación de nombre Microsoft. El PC tuvo un éxito instantáneo y dio legitimidad a la computadora de sobremesa como producto de consumo en la mente de la comunidad empresarial. Hoy día, se emplea ampliamente el término *PC* para hacer referencia a todas esas máquinas de diversos fabricantes cuyo diseño ha evolucionado a partir de la computadora personal inicial de IBM y la mayoría de ellas se ponen en el mercado con el software de Microsoft. En ocasiones, sin embargo, el término *PC* se utiliza de manera intercambiable con los términos genéricos ***computadora de sobremesa*** (***desktop***) o ***computadora portátil*** (***laptop***).

A medida que el siglo xx se aproximaba a su final, la capacidad de conectar computadoras individuales en un sistema mundial denominado **Internet** estaba revolucionando las comunicaciones. En este contexto, Tim Berners-Lee (un científico británico) propuso un sistema mediante el que podían enlazarse entre sí documentos almacenados en computadoras distribuidas por toda Internet, generando un laberinto de información enlazada que se denominó **World Wide Web** (y que a menudo se abrevia con el nombre de "Web"). Para hacer accesible la información en la Web, se desarrollaron sistemas software, conocidos como **motores de búsqueda**, los cuales permiten "explorar" la Web, "clasificar" las páginas encontradas y luego utilizar los resultados para ayudar a los usuarios que estén investigando acerca de determinados temas. Los principales actores en este campo son Google, Yahoo y Microsoft. Estas empresas continúan ampliando sus actividades relacionadas con la Web, a menudo en determinadas direcciones que desafían incluso a nuestra forma tradicional de pensar.

Al mismo tiempo que las computadoras de sobremesa (y las más modernas computadoras portátiles) comenzaban a tener aceptación y a ser utilizadas en entornos domésticos, continuaban produciéndose avances en la miniaturización de las máquinas de computación. Hoy día, hay integradas diminutas computadoras en diversos tipos de dispositivos. Por ejemplo, los automóviles contienen actualmente pequeñas computadoras en las que se ejecutan sistemas de navegación, GPS (*Global Positioning System*, Sistema de posicionamiento global), que monitorizan el funcionamiento del motor y que permiten

Google

Fundada en 1998, Google Inc. se ha convertido en una de las compañías tecnológicas más famosas del mundo. Millones de personas utilizan su servicio principal, el motor de búsqueda Google, para buscar documentos en la World Wide Web. Además, Google proporciona un servicio de correo electrónico (denominado Gmail), un servicio de compartición de vídeos basado en Internet (denominado YouTube) y una pléyade de otros servicios Internet (incluyendo Google Maps, Google Calendar, Google Earth, Google Books y Google Translate).

Sin embargo, además de ser un ejemplo notorio del espíritu emprendedor, Google también proporciona ejemplos de cómo la expansión de la tecnología plantea desafíos a nuestra sociedad. Por ejemplo, el motor de búsqueda de Google ha planteado una serie de cuestiones relativas al grado con el que una empresa internacional debe cumplir con los deseos de los gobiernos individuales; YouTube ha planteado cuestiones relativas a hasta qué punto puede una compañía ser responsable de la información que otras personas distribuyen a través de sus servicios, así como acerca del grado con el que dicha empresa pueda reclamar la propiedad de dicha información. Google Books ha planteado problemas relativos al ámbito y limitaciones de la propiedad intelectual y Google Maps ha sido acusada de violar el derecho a la intimidad.

emplear servicios de control por voz de los sistemas de comunicación telefónica y de audio del automóvil.

Quizá la aplicación más revolucionaria en el proceso de miniaturización de las computadoras es la que hace posible las cada vez mayores capacidades de los teléfonos portátiles. Ciertamente, lo que recientemente era tan solo un teléfono ha evolucionado hasta convertirse en una pequeña computadora de mano de uso general que se denomina **teléfono inteligente** (*smartphone*) y en la que la telefonía es solo una entre muchas otras aplicaciones. Estos "teléfonos" están equipados con una amplia gama de sensores e interfaces, incluyendo cámaras, micrófonos, brújulas, pantallas táctiles, acelerómetros (para detectar la orientación y el movimiento del teléfono) y una serie de tecnologías inalámbricas para comunicarse con otros teléfonos inteligentes y con computadoras. El potencial de este tipo de tecnología es enorme. De hecho, muchas personas sostienen que el teléfono inteligente tendrá un mayor efecto sobre la sociedad que el propio PC.

La miniaturización de las computadoras y sus cada vez mayores capacidades han situado a la tecnología de la computación en la vanguardia de nuestra sociedad actual. La tecnología de computadoras es tan prevalente hoy día, que la familiaridad con ella es fundamental para poder ser un miembro de pleno derecho de nuestra sociedad moderna. La tecnología de la computación ha alterado la capacidad de los gobiernos para ejercer control sobre los ciudadanos; ha tenido un enorme impacto en la globalización de la economía; ha conducido a extraordinarios avances en el campo de la investigación científica; ha revolucionado el funcionamiento de los sistemas de recopilación y almacenamiento de datos y de las aplicaciones que los utilizan; ha proporcionado nuevas formas de comunicación e interacción a las personas y ha planteado diversos desafíos al estatus actualmente vigente en nuestra sociedad. El resul-

tado es una proliferación de campos que giran alrededor de las Ciencias de la computación, cada uno de los cuales es hoy día un campo de estudio significativo en sí mismo. Además, al igual que sucede con la ingeniería mecánica y la física, a menudo es difícil trazar una línea divisoria entre esos campos y la propia ciencia de la computación. Por tanto, para adoptar la perspectiva adecuada, no solo vamos a cubrir en nuestro estudio aquellos temas que son fundamentales en el campo de las Ciencias de la computación, sino que también exploraremos diversas disciplinas que tratan tanto con las aplicaciones como con las consecuencias de esas ciencias. De hecho, una introducción a las Ciencias de la computación representa necesariamente un esfuerzo interdisciplinar.

0.3 La ciencia de los algoritmos

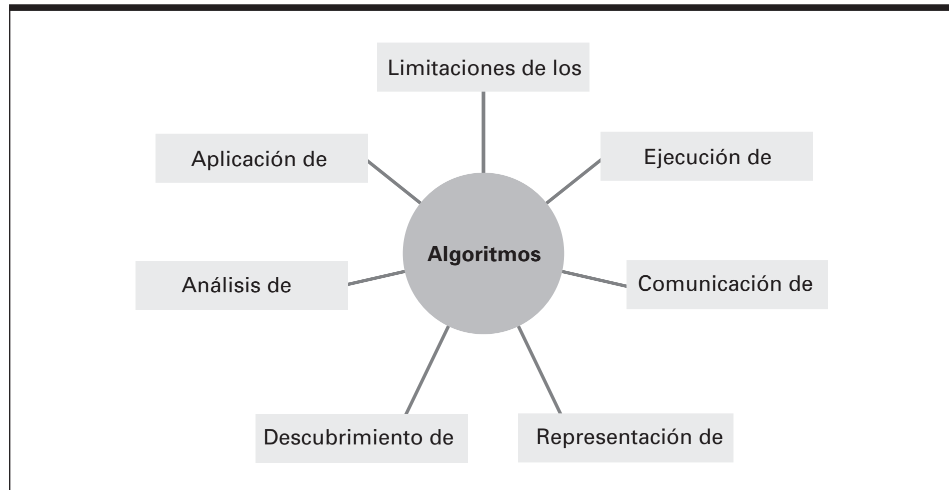
En las primeras máquinas de computación, la complejidad de los algoritmos utilizados estaba restringida por limitaciones tales como la capacidad de almacenamiento de datos y lo intrincado y tedioso de los procedimientos de programación. Sin embargo, a medida que estas limitaciones comenzaron a desaparecer las máquinas se empezaron a aplicar a tareas cada vez mayores y más complejas. A medida que los intentos de expresar estas tareas en forma algorítmica comenzaron a plantear problemas a la capacidad de la mente humana, tuvieron que dedicarse cada vez más esfuerzos de investigación al estudio de los algoritmos y al proceso de programación.

Fue en este contexto en el que el trabajo teórico de los matemáticos empezó a dar sus frutos. Como consecuencia del teorema de incompletitud de Gödel, los matemáticos ya habían estado investigando cuestiones relativas a los procesos algorítmicos que los avances de la tecnología estaban entonces planteando. Con ese bagaje, el escenario estaba dispuesto para la aparición de una nueva disciplina conocida con el nombre de ***Ciencias de la computación.***

Actualmente, las Ciencias de la computación se han consolidado como ciencia de los algoritmos. El ámbito de esta ciencia es muy amplio, abarcando campos tan diversos como las matemáticas, la ingeniería, la psicología, la biología, la administración empresarial y la lingüística. De hecho, los investigadores que trabajan en diferentes ramas de las Ciencias de computación pueden tener definiciones muy distintas acerca de dichas ciencias. Por ejemplo, alguien que se dedique a investigar en el campo de la arquitectura de computadoras podría centrarse en la tarea de miniaturización de los circuitos y ver, por tanto, las Ciencias de la computación como una serie de avances tecnológicos junto con una aplicación de los mismos. Sin embargo, alguien que investigue el campo de los sistemas de bases de datos podría ver las Ciencias de la computación como una manera de buscar la forma de hacer más útiles los sistemas de información. Y un investigador en el campo de la inteligencia artificial podría considerar las Ciencias de la computación como el estudio de la inteligencia y del comportamiento inteligente.

Por tanto, una introducción a las Ciencias de la computación debe incluir una diversidad de temas, que es la tarea a la que vamos a dedicarnos en los siguientes capítulos. En cada caso, los objetivos serán presentar las ideas fundamentales de cada campo, los temas actuales de investigación y algunas de las

Figura 0.5 El papel central de los algoritmos en las Ciencias de la computación.



técnicas que se están aplicando para hacer avanzar los conocimientos dentro de esa área. Con esta diversidad de temas será fácil perder de vista la imagen más general. Por tanto, vamos a hacer una pausa para ordenar nuestra mente, identificando algunas cuestiones que nos permitan centrar el estudio de las Ciencias de la computación.

- ¿Qué problemas pueden resolverse mediante procesos algorítmicos?
- ¿Cómo puede facilitarse el descubrimiento de algoritmos?
- ¿Cómo pueden mejorarse las técnicas de representación y comunicación de algoritmos?
- ¿Cómo pueden analizarse y compararse las características de los diferentes algoritmos?
- ¿Cómo pueden utilizarse los algoritmos para tratar la información?
- ¿Cómo pueden aplicarse los algoritmos para generar un comportamiento inteligente?
- ¿Cómo afecta a la sociedad la aplicación de algoritmos?

Observe que el tema común a todas estas cuestiones es el estudio de los algoritmos (Figura 0.5).

0.4 Abstracción

El concepto de abstracción impregna hasta tal punto el estudio de las Ciencias de la computación y el diseño de los sistemas de computadoras, que nos vemos obligados a tenerlo en cuenta en este capítulo preliminar. El término **abstracción**, tal como lo estamos utilizando aquí, hace referencia a la distinción entre las propiedades externas de una entidad y los detalles de la composición interna de la misma. Es la abstracción lo que nos permite ignorar los detalles internos de un dispositivo complejo tal como una computadora, un

automóvil o un microondas y emplearlo como una única unidad comprensible. Además, es gracias a la abstracción que se pueden diseñar y fabricar dichos sistemas complejos. Las computadoras, los automóviles y los hornos microondas se construyen a partir de componentes, cada uno de los cuales está a su vez construido a partir de otros componentes más pequeños. Cada componente representa un nivel de abstracción, en el sentido de que el uso de ese componente está aislado de los detalles de la composición interna del componente.

Es, por tanto, gracias a que aplicamos la abstracción que somos capaces de construir, analizar y gestionar sistemas de computadoras grandes y complejos que nos resultarían inmanejables si los contempláramos en su totalidad con un nivel detallado. En cada nivel de abstracción, contemplamos el sistema en términos de una serie de componentes, denominados **herramientas abstractas**, cuya composición interna ignoramos. Esto nos permite concentrarnos en cómo interactúa con los restantes componentes del mismo nivel y cómo el conjunto de todos los componentes forma un componente de nivel superior. De este modo, somos capaces de entender la parte del sistema que sea relevante para la parte del sistema que tengamos entre manos, en lugar de perdernos en un océano de detalles.

Conviene recalcar que el concepto de abstracción no está limitado a los campos de la ciencia y la tecnología. Se trata de una técnica importante de simplificación, gracias a la cual nuestra sociedad ha creado un estilo de vida que sería imposible si no utilizáramos ese concepto. Pocos de nosotros comprendemos cómo se implementan en realidad los diversos aparatos, productos y servicios que tan útiles nos son en la vida cotidiana. Ingerimos alimentos y vestimos prendas de ropa que no seríamos capaces de producir por nosotros mismos. Utilizamos dispositivos eléctricos y sistemas de comunicación sin entender la tecnología subyacente. Empleamos los servicios de otras personas sin conocer los detalles de sus respectivas profesiones. Con cada nuevo avance, una pequeña parte de la sociedad decide especializarse en su implementación mientras que el resto de nosotros aprendemos a utilizar los resultados en forma de herramientas abstractas. De esta forma, el conjunto de herramientas abstractas de la sociedad se va expandiendo cada vez más y la capacidad de progresar de la sociedad se incrementa.

La abstracción será un tema recurrente a lo largo del libro. Veremos que los equipos de computación se construyen en una serie de niveles de herramientas abstractas. También veremos que el desarrollo de sistemas software de gran envergadura se lleva a cabo de una forma modular, de manera que cada módulo se emplea como una herramienta abstracta a la hora formar otros módulos de mayor complejidad. Además, la abstracción desempeña un papel importante en la tarea de hacer progresar a las propias Ciencias de la computación, haciendo a los investigadores centrar su atención en áreas concretas dentro un campo complejo. De hecho, la organización de este libro refleja esta característica de las Ciencias de la computación. Cada capítulo, que está centrado en un área concreta de dichas ciencias, suele ser sorprendentemente independiente de los restantes capítulos, a pesar de lo cual todos los capítulos forman una panorámica bastante completa de lo que constituye un amplio campo de estudio.

0.5 Un resumen de nuestro estudio

Este texto sigue un enfoque de abajo-arriba para el estudio de las Ciencias de la computación, comenzando con temas tan concretos como el hardware de computadoras y ascendiendo hacia los temas más abstractos como la complejidad algorítmica y la computabilidad. El resultado es que nuestro estudio sigue la técnica de construir herramientas abstractas cada vez más complejas a medida que se amplía nuestra comprensión de este campo de la ciencia.

Comenzamos considerando los temas de diseño y construcción de máquinas para la ejecución de algoritmos. En el Capítulo 1 (Almacenamiento de datos) examinamos cómo se codifica y almacena la información en las computadoras modernas y en el Capítulo 2 (Tratamiento de datos) investigamos el funcionamiento interno básico de una computadora sencilla. Aunque parte de este estudio implica un cierto contacto con la tecnología, el tema general es independiente de esta. Es decir, cuestiones tales como el diseño de circuitos digitales, los sistemas de codificación y compresión de datos y la arquitecura de computadoras son relevantes en un amplio rango de tecnologías y continuarán siendo relevantes independientemente de la dirección que la tecnología futura siga.

En el Capítulo 3 (Sistemas operativos) estudiamos el software que permite controlar el funcionamiento global de una computadora. Este software se denomina sistema operativo. El sistema operativo de una computadora controla la interfaz entre la máquina y el mundo exterior, protegiendo a la máquina y a los datos almacenados en ella frente a accesos no autorizados, permitiendo al usuario de una computadora solicitar la ejecución de diversos programas y coordinando las actividades internas necesarias para satisfacer las solicitudes del usuario.

En el Capítulo 4 (Redes e Internet) estudiamos cómo se conectan entre sí las computadoras para formar redes de computadoras y cómo esas redes se conectan para formar interredes. Dicho estudio nos conducirá a temas tales como los protocolos de red, la estructura y funcionamiento interno de Internet, la World Wide Web y numerosas cuestiones relativas a la seguridad.

El Capítulo 5 (Algoritmos) presenta el estudio de los algoritmos desde una perspectiva más formal. Investigaremos cómo se descubren los algoritmos, identificaremos varias estructuras algorítmicas fundamentales, desarrollaremos técnicas elementales para la representación de algoritmos y presentaremos los temas relativos a la eficiencia y corrección de los algoritmos.

En el Capítulo 6 (Lenguajes de programación) analizaremos el tema de la representación de algoritmos y el proceso de desarrollo de programas. Aquí veremos que la búsqueda de técnicas mejores de programación ha conducido a diversas tecnologías de programación o paradigmas, cada uno con su propio conjunto de lenguajes de programación. Investigaremos estos paradigmas y lenguajes y tendremos también en cuenta las cuestiones de traducción de lenguajes y gramáticas.

El Capítulo 7 (Ingeniería del software) introduce la rama de las Ciencias de la computación conocida con el nombre de ingeniería del software, que se ocupa de los problemas con que nos encontramos a la hora de desarrollar sistemas software de gran complejidad. El problema subyacente es que el diseño de sistemas software de gran envergadura es una tarea muy compleja, que abarca cuestiones que caen fuera del campo de la ingeniería tradicional. Por

ello, el tema de la ingeniería software se ha convertido en un campo importante de investigación dentro de las Ciencias de la computación, en el que se emplean resultados de campos tan diversos como la ingeniería, la gestión de proyectos, la gestión de personal, el diseño de lenguajes de programación e incluso la arquitectura.

En los dos siguientes capítulos examinamos la forma en que pueden organizarse los datos en una computadora. En el Capítulo 8 (Abstracciones de datos) presentamos las técnicas tradicionalmente empleadas para organizar los datos en la memoria principal de una computadora y luego examinaremos la evolución de la abstracción de datos, desde el concepto de primitivas hasta las técnicas actuales de orientación a objetos. En el Capítulo 9 (Sistemas de bases de datos) consideramos los métodos tradicionalmente empleados para organizar los datos dentro de los sistemas de almacenamiento masivo de una computadora. Investigamos también en este capítulo cómo se implementan los sistemas de base de datos extremadamente grandes y complejos.

En el Capítulo 10 (Gráficos por computadora) exploramos el tema de los gráficos y la animación, un campo que se ocupa de la creación de mundos virtuales y de la generación de imágenes de los mismos. Basándose en los avances de otras áreas más tradicionales de las Ciencias de la computación, como la arquitectura de máquinas, el diseño de algoritmos, las estructuras de datos y la ingeniería software, la disciplina de los gráficos y la animación ha experimentado un progreso significativo y ha terminado por florecer en un nuevo campo excitante y dinámico. Además, este campo constituye un ejemplo extraordinario de cómo los distintos componentes de las Ciencias de la computación se combinan con otras disciplinas como la física, el arte y la fotografía para producir resultados sorprendentes.

En el Capítulo 11 (Inteligencia artificial) aprenderemos que para poder desarrollar máquinas más útiles, las Ciencias de la computación han centrado su atención en el estudio de la inteligencia humana, en busca de claves que permitan continuar progresando. La esperanza es que entendiendo cómo razonan y perciben nuestras propias mentes, los investigadores serán capaces de diseñar algoritmos que imiten dichos procesos y transferir dichas capacidades a las máquinas. El resultado es el área de las Ciencias de la computación conocida con el nombre de inteligencia artificial, que depende en gran medida de las investigaciones realizadas en áreas tales como la psicología, la biología y la lingüística.

Cerraremos nuestro estudio con el Capítulo 12 (Teoría de la computación), investigando los fundamentos teóricos de las Ciencias de la computación, un tema que nos permitirá comprender las limitaciones de los algoritmos (y por tanto de las máquinas). En este capítulo identificaremos algunos problemas que no pueden resolverse algorítmicamente (y por tanto caen fuera de las capacidades de las máquinas) y veremos también que las soluciones a muchos otros problemas requieren una cantidad de tiempo o espacio tan enorme que también son irresolubles desde un punto de vista práctico. Así, gracias a este estudio podremos ver cuál es el ámbito de los sistemas algorítmicos y cuáles son sus limitaciones.

En cada capítulo, nuestro objetivo es explorar cada tema concreto hasta una profundidad que nos permita comprender verdaderamente la materia. Queremos desarrollar en el lector un conocimiento práctico de las Ciencias de

la computación, un conocimiento que le permita entender la sociedad técnica en la que vivimos y que le proporcione una base a partir de la cual pueda seguir aprendiendo por sí mismo a medida que la ciencia y la tecnología avancen.

0.6 Repercusiones sociales

El progreso en el campo de las ciencias de la computación está haciendo que se difuminen muchas distinciones en las que nuestra sociedad ha basado sus decisiones en el pasado, y está poniendo en cuestión muchos de los principios largamente sostenidos en nuestra sociedad. En el campo de las leyes, genera cuestiones relativas al grado con el que se puede ser poseedor de la propiedad intelectual y también en relación a los derechos y responsabilidades que acompañan dicha posesión. En el campo de la ética, genera numerosas opciones que desafían los principios tradicionales en los que se basa el comportamiento social. En el campo de la acción de gobierno, genera debates relativos al grado con el que habría que regular la tecnología informática y sus aplicaciones. En el terreno filosófico, genera un debate entre la presencia del comportamiento inteligente y la presencia de la propia inteligencia. Y en toda la sociedad genera disputas relativas a si las nuevas aplicaciones representan nuevas libertades o nuevos controles.

Aunque no forman parte de las Ciencias de la computación, estos temas son importantes para aquellos que estén pensando en desarrollar su carrera en el campo de la computación o en algún campo relacionado. Los avances científicos han encontrado en ocasiones aplicaciones controvertidas, provocando un serio descontento a los investigadores que en ellos participaron. Además, una carrera llena de éxitos profesionales puede descarrilar rápidamente debido a una equivocación ética.

La capacidad de tratar con los dilemas planteados por los avances en la tecnología de computadoras también es importante para aquellos que no están directamente involucrados en esos avances. De hecho, la tecnología está permeando la sociedad de forma tan rápida que son pocas las personas que no se ven afectadas por los avances tecnológicos, si es que hay alguna.

Este texto proporciona los conocimientos técnicos necesarios para analizar de forma racional los dilemas generados por las Ciencias de la computación. Sin embargo, el conocimiento técnico de las disciplinas científicas no proporciona por sí solo soluciones a todas las cuestiones planteadas. Teniendo esto en cuenta, el libro incluye varias secciones dedicadas a cuestiones sociales, éticas y legales. Entre estas se incluyen las relativas a la seguridad, la propiedad del software y responsabilidades legales, el impacto social de la tecnología de bases de datos y las consecuencias de los avances en el campo de la inteligencia artificial.

Además, a menudo no existe ninguna respuesta definitivamente correcta a un problema y muchas soluciones válidas suelen ser compromisos entre puntos de vista opuestos (y quizá igualmente válidos). Encontrar soluciones en estos casos a menudo requiere la capacidad de escuchar, de entender otros puntos de vista, de llevar a cabo un debate racional y de modificar la propia opinión a medida que se adquieren nuevas formas de ver las cosas. Por ello, cada capítulo de este texto termina con una colección de preguntas bajo el epí-

grafe “Cuestiones sociales” que investigan las relaciones entre las Ciencias de la computación y la sociedad. No se trata necesariamente de preguntas que haya que responder. Más bien, son cuestiones que conviene plantearse. En muchos casos, una respuesta que puede parecer obvia a primera vista, dejará de satisfacernos a medida que vayamos explorando las alternativas. Por resumir, el propósito de estas cuestiones no es conducir al lector a una respuesta “correcta”, sino más bien ayudarle a ser consciente de todas las dimensiones del problema, incluyendo ser consciente de los diversos intereses en una cuestión, de las alternativas y de las consecuencias a corto y largo plazo de dichas alternativas.

Vamos a cerrar esta sección presentando algunos de los planteamientos éticos realizados por los filósofos, en su búsqueda de teorías fundamentales que permitan enunciar una serie de principios que sirvan de guía para nuestras decisiones y nuestro comportamiento. La mayoría de estas teorías pueden clasificarse en una serie de grupos: ética basada en las consecuencias, ética basada en el deber, ética basada en los contratos y ética basada en el carácter. El lector puede emplear dichas teorías como forma de analizar las cuestiones éticas planteadas en el texto. En particular, podrá ver que las diferentes teorías conducen a conclusiones contrapuestas, exponiendo así toda una serie de alternativas ocultas.

La ética basada en las consecuencias trata de analizar las cuestiones basándose en las consecuencias de las distintas opciones. Un ejemplo sobresaliente es el utilitarismo que propone que la decisión o acción “correcta” es aquella que conduce a un mayor beneficio para una mayor parte de la sociedad. A primera vista, el utilitarismo parece ser una forma adecuada de resolver los dilemas éticos. Pero, si no se matiza, el utilitarismo conduce a numerosas conclusiones inaceptables. Por ejemplo, permitiría que la mayoría de la sociedad esclavizara a una pequeña minoría. Además, muchos pensadores argumentan que el análisis de las teorías éticas basadas en las consecuencias, que pone el énfasis inherentemente en esas consecuencias, tiende a contemplar a los seres humanos como si fueran un simple medio para un fin, en lugar de ser individuos que merecen la pena por sí mismos. Esto constituye, según esos pensadores, un fallo fundamental en todas las teorías éticas basadas en las consecuencias.

En contraposición a la ética basada en las consecuencias, la ética basada en el deber no considera las consecuencias de las decisiones y acciones, sino que propone en su lugar que los miembros de la sociedad tienen ciertos deberes y obligaciones intrínsecos que forman, a su vez, la base con la que resolver las cuestiones éticas. Por ejemplo, si uno acepta la obligación de respetar los derechos de otros, entonces es preciso rechazar la esclavitud independientemente de cuáles sean sus consecuencias. Por otro lado, los oponentes de la ética basada en el deber argumentan que este tipo de ética no consigue proporcionar soluciones a aquellos problemas en los que existen deberes en conflicto. ¿Debemos por ejemplo decir la verdad incluso si el hacerlo destruye la autoconfianza de un colega? ¿Debe una nación defenderse en una guerra aun cuando las consiguientes batallas lleven a la muerte a muchos de sus ciuda-danos?

La teoría ética basada en el contrato social comienza por imaginar una sociedad en la que no exista ninguna base ética. En este “estado natural” de las cosas, vale todo, se trata de una situación en la que cada individuo debe mirar por sí mismo y estar constantemente en guardia frente a las posibles agresiones

de los otros. En esas circunstancias, la teoría ética basada en el contrato social propone que los miembros de la sociedad desarrollen contratos entre ellos. Por ejemplo, yo no te robaré a tí si tú no me robas a mí. A su vez, "estos contratos" se convierten en la base para determinar el comportamiento ético. Observe que la teoría ética basada en el contrato social proporciona una motivación para el comportamiento ético, necesitamos obedecer los "contratos éticos" porque en caso contrario nuestra vida sería más desagradable. Sin embargo, los oponentes de la teoría ética basada en el contrato social argumentan que dicha teoría no proporciona una base suficiente para resolver los dilemas éticos, ya que solo suministra una guía para aquellos en los que se hayan establecido los pertinentes contratos. (Yo puedo comportarme de cualquier forma que desee en aquellas situaciones que no estén cubiertas por un contrato existente.) En particular, las nuevas tecnologías pueden constituir un territorio inexplorado en el que los contratos éticos existentes pueden no ser de aplicación.

La ética basada en el carácter (en ocasiones denominada ética de la virtud), que fue planteada por Platón y Aristóteles, argumenta que el "comportamiento correcto" no es el resultado de la aplicación de reglas identificables, sino una simple consecuencia natural del "buen carácter". Mientras que la ética basada en las consecuencias, la ética basada en el deber y la ética basada en el contrato proponen que las personas resuelvan los dilemas éticos preguntándose "¿Cuáles son las consecuencias?", "¿Cuál es mi deber?" o "¿Qué contratos me obligan?", la ética basada en el carácter propone que los dilemas se resuelvan preguntándose "¿Quién quiero ser?". Por tanto, el comportamiento adecuado se exhibe fomentando un buen carácter, que es típicamente el resultado de una crianza correcta y del desarrollo de hábitos virtuosos.

Es una ética basada en el carácter la que se emplea normalmente para fundamentar el enfoque adoptado a la hora de "enseñar" ética a los profesionales de diversos campos. En lugar de presentar teorías éticas específicas, el enfoque consiste en introducir casos de estudio que ponen de manifiesto diversas cuestiones éticas dentro del área de experiencia de los profesionales en cuestión. Entonces, analizando los pros y los contras en cada caso, los profesionales pasan a ser más conscientes, más sensibles y más conocedores de los peligros que les acechan en sus vidas profesionales, forjando así un poco más su carácter. Este es el espíritu con el que se presentan al final de cada capítulo las cuestiones concernientes a los temas sociales.

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Generalmente se acepta la premisa de que nuestra sociedad es ***diferente*** de lo que sería si no se hubiera producido la revolución informática. ¿Es nuestra sociedad ***mejor*** de lo que hubiera sido sin dicha revolución? ¿Es nuestra sociedad peor? ¿Sería diferente su respuesta si su posición dentro de la sociedad fuera distinta?

2. ¿Es aceptable participar en la sociedad técnica actual sin hacer un esfuerzo por comprender los fundamentos de dichas tecnologías? Por ejemplo, ¿cree que los ciudadanos de un país democrático, cuyos votos determinan a menudo cómo se apoyará y se utilizará la tecnología, tienen la obligación de tratar de entender dicha tecnología? ¿Depende su respuesta de la tecnología que se esté considerando? Por ejemplo, ¿sería su respuesta igual a la hora de considerar la tecnología nuclear que a la hora de considerar la tecnología de computadoras?
3. Utilizando dinero en efectivo en las transacciones financieras, las personas han dispuesto tradicionalmente de la opción de gestionar sus asuntos financieros sin que les cobraran comisiones de servicio. Sin embargo, a medida que se va automatizando una parte cada vez mayor de nuestra economía, las instituciones financieras aplican comisiones de servicio para el acceso a estos sistemas automatizados. ¿Cree que existe un punto en que dichas comisiones restringen de manera no equitativa el acceso de las personas a la economía? Por ejemplo, suponga que un empresario paga a sus empleados exclusivamente mediante un cheque y que todas las entidades financieras aplican una comisión de servicio a las operaciones de cobro y depósito de cheques. ¿Implicaría esto un trato injusto hacia los empleados? ¿Qué sucedería si un empresario insistiera en pagar únicamente mediante transferencia a una cuenta corriente?
4. En el contexto de la televisión interactiva, ¿en qué grado debería permitirse a las empresas recopilar información de los niños (quizá mediante un formato de juego interactivo)? Por ejemplo, ¿debería permitirse a las empresas obtener un informe de los niños sobre los hábitos de compra de sus padres? ¿Debería permitírseles obtener información acerca del propio niño?
5. ¿Hasta qué punto debería un gobierno regular la tecnología de computadoras y sus aplicaciones? Considere, por ejemplo, los problemas planteados en las Cuestiones 3 y 4. ¿Qué razones son las que justificarían la regulación gubernamental?
6. ¿En qué grado pueden afectar a las generaciones futuras nuestras decisiones concernientes a la tecnología en general y a la tecnología de las computadoras en particular?
7. A medida que avanza la tecnología, nuestro sistema educativo se ve enfrentado al desafío de reconsiderar el nivel de abstracción con el que se presentan los temas. Muchas de las cuestiones que se plantean son del tipo de si una capacidad sigue siendo necesaria o si, por el contrario, debería permitirse a los estudiantes utilizar una herramienta abstracta. Por ejemplo, a los estudiantes de trigonometría ya no se les enseña a calcular los valores de las funciones trigonométricas utilizando tablas. En lugar de ello, emplean calculadoras como herramientas abstractas para determinar dichos valores. Algunas personas argumentan que también debería prescindirse de enseñar el algoritmo de división sustituyéndolo por una abstracción. ¿Qué otros temas se ven afectados por controversias similares? ¿Cree que los procesadores de texto modernos eliminan la necesidad de dominar la ortografía? ¿Cree que el uso de la tecnología de vídeo hará que desaparezca algún día la necesidad de leer?

8. El concepto de biblioteca pública se basa, en gran medida, en la premisa de que todos los ciudadanos de una democracia deben tener acceso a la información. A medida que se disemina y almacena cada vez más información mediante tecnología de computadoras, ¿cree que el acceso a esta tecnología se convierte en un derecho de todo individuo? Si esto es así, ¿deberían ser las bibliotecas públicas el canal a través del cual se proporcionara dicho acceso?
9. ¿Qué problemas éticos surgen en una sociedad que depende del uso de herramientas abstractas? ¿Existen casos en los que no resulte ético utilizar un producto o servicio sin entender cómo funciona? ¿Y existen casos en los que no resulta ético utilizar un producto o servicio sin saber cómo se produce o suministra?, ¿o sin entender cuáles son las consecuencias de su uso?
10. A medida que nuestra sociedad se automatiza cada vez, resulta más fácil para los gobiernos vigilar las actividades de sus ciudadanos. ¿Cree que esto es bueno o es malo?
11. ¿Qué tecnologías imaginadas por George Orwell (Eric Blair) en su novela *1984* se han hecho realidad? ¿Se están utilizando en la forma en que Orwell predijo?
12. Si existiera una máquina del tiempo, ¿en qué periodo de la historia querría vivir? ¿Hay alguna tecnología actual que le gustaría llevar consigo? ¿Podría llevar esas tecnologías consigo sin llevar al mismo tiempo otras tecnologías? ¿Hasta que punto puede separarse una tecnología de otra? ¿Es coherente protestar contra el calentamiento global y, sin embargo, aceptar los nuevos tratamientos médicos?
13. Suponga que su trabajo le obligara a vivir en un país con otra cultura. ¿Debería continuar aplicando los principios éticos de su cultura nativa o adoptar la ética de la cultura de su país anfitrión? ¿Depende su respuesta de si los temas planteados afectan a la forma de vestir o a los derechos humanos? ¿Qué estándares éticos deberían prevalecer si continúa residiendo en su país nativo pero tiene que hacer negocios con un país extranjero, con una cultura diferente?
14. ¿Cree que la sociedad se ha hecho demasiado dependiente de las aplicaciones informáticas para el comercio, las comunicaciones o las relaciones sociales? Por ejemplo, ¿cuáles serían las consecuencias de una interrupción prolongada en el servicio de Internet y/o de telefonía móvil?
15. La mayor parte de los teléfonos inteligentes son capaces de identificar la ubicación del teléfono por medio del sistema GPS. Esto permite a las aplicaciones proporcionar información específica de la ubicación (como por ejemplo las noticias locales, el pronóstico del tiempo en esa localidad o información acerca de las empresas situadas en las proximidades), basándose en la ubicación actual del teléfono. Sin embargo, dichas capacidades GPS también pueden permitir a otras aplicaciones informar acerca de la ubicación del teléfono a otras personas. ¿Es eso bueno? ¿Qué posibles abusos podrían darse si alguien conociera la ubicación del teléfono (y por tanto nuestra ubicación)?

16. Teniendo en cuenta su respuesta inicial a las cuestiones anteriores, ¿qué teoría ética de las presentadas en la Sección 0.6, le parece más correcta?

Lecturas adicionales

Goldstine, J. J. *The Computer from Pascal to von Neumann*. Princeton: Princeton University Press, 1972.

Kizza, J. M. *Ethical and Social Issues in the Information Age*, 3ª ed. Londres: Springer-Verlag, 2007.

Mollenhoff, C. R. *Atanasoff: Forgotten Father of the Computer*. Ames: Iowa State University Press, 1988.

Neumann, P. G. *Computer Related Risks*. Boston, MA: Addison-Wesley, 1995.

Ni, L. *Smart Phone and Next Generation Mobile Computing.* San Francisco: Morgan Kaufmann, 2006.

Quinn, M. J. *Ethics for the Information Age*, 2ª ed. Boston, MA: Addison-Wesley, 2006.

Randell, B. *The Origins of Digital Computers*, 3ª ed. Nueva York: Springer-Verlag, 1982.

Spinello, R. A. y H. T. Tavani. *Readings in CyberEthics*, 2ª ed. Sudbury, MA: Jones and Bartlett, 2004.

Swade, D. *The Difference Engine*. Nueva York: Viking, 2000.

Tavani, H. T. *Ethics and Technology: Ethical Issues in an Age of Information and Communication Technology*, 3ª ed. Nueva York: Wiley, 2011.

Woolley, B. *The Bride of Science, Romance, Reason, and Byron's Daughter*. Nueva York: McGraw-Hill, 1999.

CAPÍTULO

1

Almacenamiento de datos

En este capítulo vamos a considerar los temas asociados con la representación y el almacenamiento de datos dentro de una computadora. Entre los tipos de datos que tomaremos en consideración se incluyen los textos, los valores numéricos, las imágenes, el audio y el vídeo. Buena parte de la información contenida en este capítulo es también relevante en otros campos distintos de la computación tradicional, como la fotografía digital, la grabación y reproducción de audio/vídeo y las comunicaciones a larga distancia.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

Comenzaremos nuestro estudio de las Ciencias de la computación analizando cómo se codifica y almacena la información dentro de las computadoras. El primer paso consistirá en exponer los datos básicos acerca de los dispositivos de almacenamiento de datos de una computadora y luego considerar cómo se codifica la información para almacenarla en dichos sistemas. Exploraremos las ramificaciones de los sistemas de almacenamiento de datos actuales y veremos cómo se emplean técnicas tales como la compresión de datos y el tratamiento de errores para compensar las limitaciones de dichos sistemas.

1.1 Los bits y su almacenamiento

Dentro de las computadoras actuales, la información se codifica mediante patrones de 0s y 1s. Estos dígitos se denominan **bits** (abreviatura de *binary digits,* dígitos binarios). Aunque uno podría sentirse inclinado a asociar los bits con valores numéricos, se trata realmente de símbolos cuyo significado depende de cada aplicación concreta. En ocasiones, los patrones de bits se usan para representar valores numéricos; en otros casos, representan caracteres de un alfabeto y signos de puntuación, a veces representan imágenes y otras sonidos.

Operaciones booleanas

Para entender cómo se almacenan y manipulan los bits individuales dentro de una computadora, es conveniente imaginar que el bit 0 representa el valor *falso* y que el bit 1 representa el valor *verdadero,* porque eso nos permite pensar en la manipulación de bits como si se tratara de la manipulación de valores verdadero/falso. Las funciones que permiten manipular valores verdadero/falso son las **operaciones booleanas**, en honor del matemático George Boole (1815–1864), que fue un pionero en el campo de las matemáticas conocido con el nombre de lógica. Tres de las operaciones booleanas básicas son AND, OR y XOR (OR exclusiva), como se resume en la Figura 1.1. Estas operaciones son similares a las operaciones aritméticas de multiplicación y suma, porque combinan un par de valores (los datos de entrada de la operación) para generar un tercer valor (la salida). Sin embargo, a diferencia de las operaciones matemáticas, las operaciones booleanas combinan valores de tipo verdadero/falso en lugar de valores numéricos.

La operación booleana AND está diseñada para reflejar la verdad o falsedad de un enunciado formado como combinación de dos enunciados más pequeños o simples, mediante la conjunción ***Y*** (*and*). Dichos enunciados tienen la forma genérica

P AND Q

donde *P* representa un enunciado y Q representa otro. Por ejemplo,

Kermit es una rana AND Miss Piggy es una actriz.

Las entradas de la operación AND representan la verdad o falsedad de los componentes del enunciado compuesto; la salida representa la verdad o falsedad del propio enunciado compuesto. Puesto que un enunciado de la forma *P* AND *Q* solo es cierto cuando sus dos componentes son verdaderos, podemos concluir

que 1 AND 1 debe ser 1, mientras que todos los demás casos deben producir una salida igual a 0, de acuerdo con la Figura 1.1.

De forma similar, la operación OR está basada en enunciados compuestos de la forma

P OR *Q*

donde, de nuevo, *P* representa un enunciado y *Q* representa otro. Tales enunciados son verdaderos cuando al menos uno de sus componentes es cierto, lo que concuerda con la operación OR mostrada en la Figura 1.1.

No existe ninguna conjunción en español que consiga capturar el significado de la operación XOR. XOR produce una salida igual a 1 (verdadero) cuando una de sus entradas es 1 (verdadero) y la otra es 0 (falso). Por ejemplo, un enunciado de la forma *P* XOR *Q* significa "*P* o *Q* son verdaderas, pero no ambas son verdaderas a la vez". (En pocas palabras, la operación XOR genera una salida igual a 1 cuando sus entradas son diferentes.)

La operación NOT es otra operación booleana. Se diferencia de AND, OR y XOR en que solo tiene una entrada. Su salida es la opuesta de la entrada. Si la entrada de la operación NOT es verdadera, entonces la salida es falsa, y viceversa. Por tanto, si la entrada de la operación NOT es la verdad o la falsedad del enunciado

Fozzie es un oso.

entonces la salida representará la verdad o falsedad del enunciado

Fozzie no es un oso.

Puertas y biestables

Un dispositivo que genera la salida de una operación booleana cuando se le proporcionan los valores de entrada de dicha operación se denomina **puerta**

Figura 1.1 Las operaciones booleanas AND, OR y XOR (OR exclusiva).

La operación AND

```
     0        0        1        1
AND  0   AND  1   AND  0   AND  1
------   ------   ------   ------
     0        0        0        1
```

La operación OR

```
     0        0        1        1
OR   0   OR   1   OR   0   OR   1
------   ------   ------   ------
     0        1        1        1
```

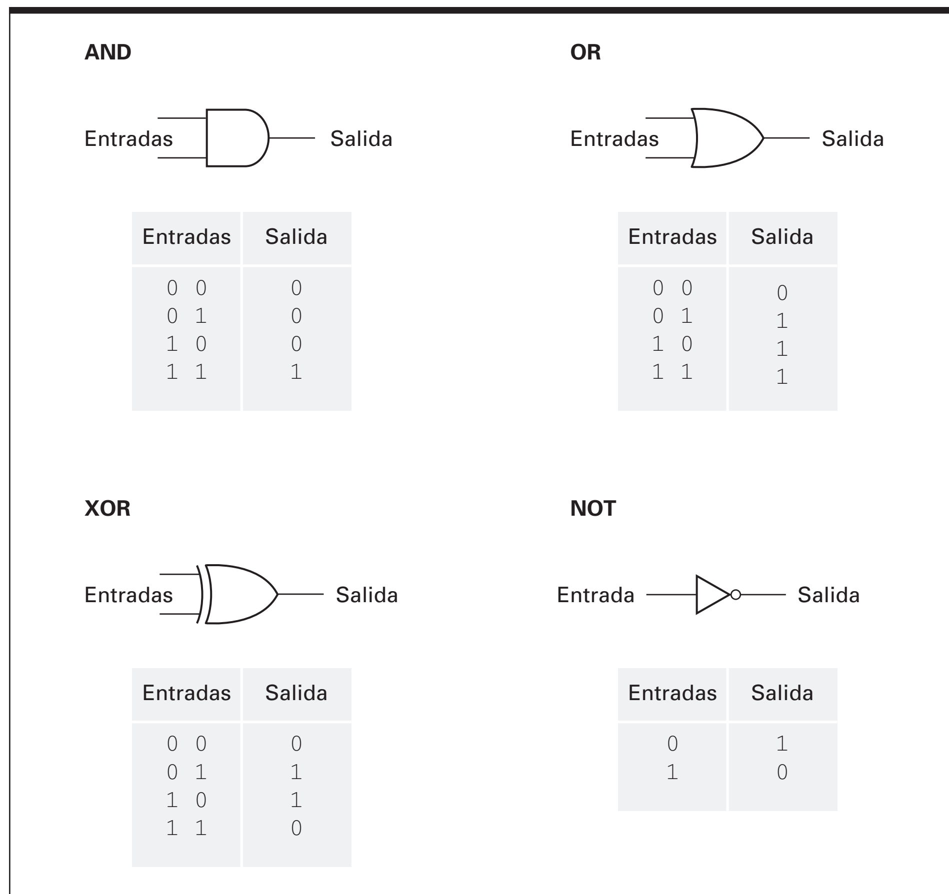
La operación XOR

```
     0        0        1        1
XOR  0   XOR  1   XOR  0   XOR  1
------   ------   ------   ------
     0        1        1        0
```

lógica. Las puertas lógicas pueden construirse mediante diversas tecnologías, como engranajes, relés y dispositivos ópticos. Dentro de las computadoras actuales, las puertas suelen implementarse con pequeños circuitos electrónicos en los que los dígitos 0 y 1 representan niveles de tensión. Sin embargo, no necesitamos preocuparnos por tales detalles. Para lo que a nosotros nos interesa, basta con representar las puertas en su forma simbólica, como se muestra en la Figura 1.2. Observe que las puertas AND, OR, XOR y NOT están representadas mediante símbolos con formas distintivas, suministrándose los valores de entrada por un lado y obteniéndose la salida por el otro.

Las puertas proporcionan los componentes a partir de los cuales se construyen las computadoras. Un paso importante en esta dirección es el que se ilustra en el circuito de la Figura 1.3. Este es un ejemplo concreto de una colección de circuitos que se conocen con el nombre de **biestables**. Un biestable es un circuito que genera un valor de salida igual a 0 o a 1, que permanece constante hasta que un pulso (un cambio temporal a un 1 que luego vuelve a 0) generado por otro circuito hace que el biestable conmute al otro valor. En otras palabras, la salida conmutará entre los dos valores bajo el control de algún estímulo

Figura 1.2 Representación simbólica de las puertas AND, OR, XOR y NOT, junto con sus valores de entrada y de salida.



AND

Entradas	Salida
0 0	0
0 1	0
1 0	0
1 1	1

OR

Entradas	Salida
0 0	0
0 1	1
1 0	1
1 1	1

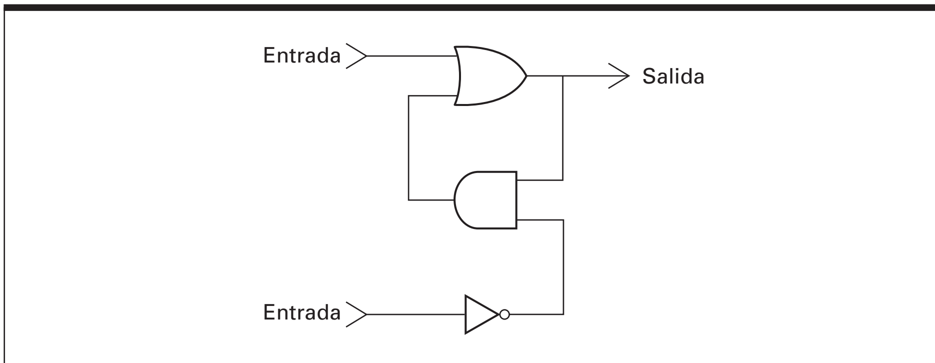
XOR

Entradas	Salida
0 0	0
0 1	1
1 0	1
1 1	0

NOT

Entradas	Salida
0	1
1	0

Figura 1.3 Un circuito biestable simple.



externo. Mientras que ambas entradas del circuito de la Figura 1.3 sigan teniendo el valor 0, la salida (ya sea 0 o 1) no variará. Sin embargo, si colocamos temporalmente un 1 en la entrada superior, forzaremos a la salida a tomar el valor 1, mientras que si colocamos temporalmente un 1 en la entrada inferior, forzaremos a la salida a ser 0.

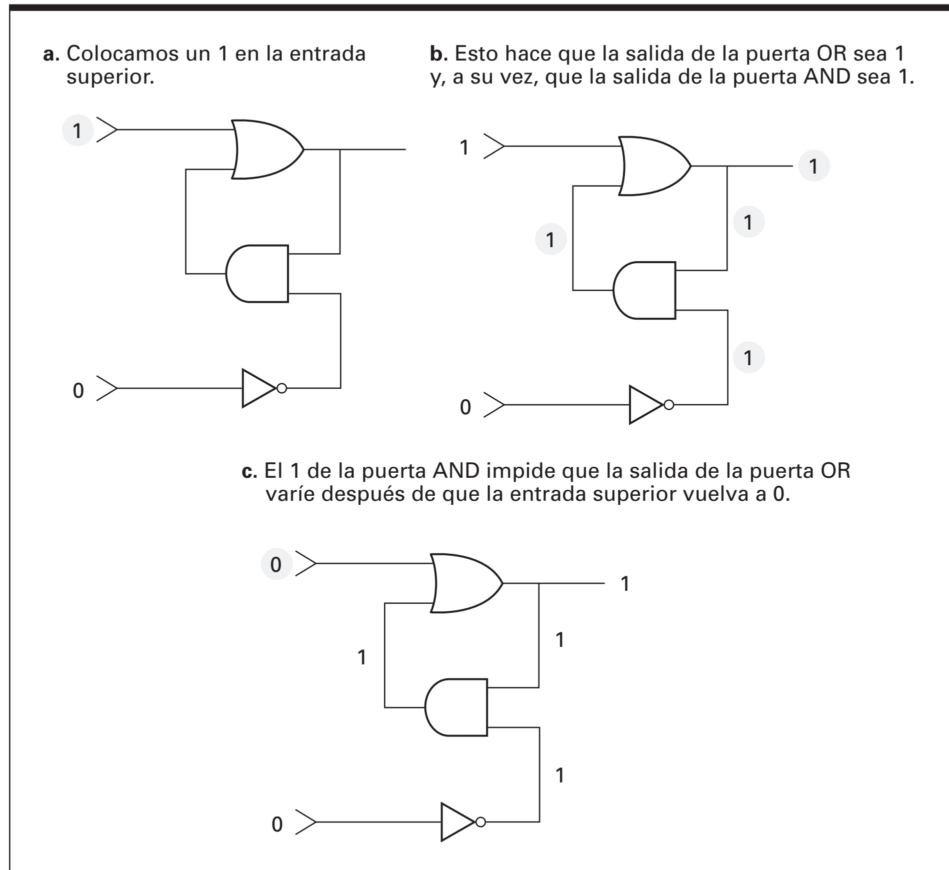
Analicemos esto con más detalle. Sin conocer la salida actual del circuito de la Figura 1.3, vamos a suponer que cambiamos la entrada superior a 1 mientras que mantenemos la entrada inferior a 0 (Figura 1.4a). Esto hará que la salida de la puerta OR sea 1, independientemente de cuál sea el valor de la otra entrada de esta puerta. A su vez, ambas entradas de la puerta AND tendrán ahora el valor 1, dado que la otra entrada a dicha puerta ya está a 1 (la salida producida por la puerta NOT cuando la entrada inferior del biestable tiene un valor 0). La salida de la puerta AND será entonces 1, lo que implica que la segunda entrada a la puerta OR estará ahora a 1 (Figura 1.4b). Esto garantiza que la salida de la puerta OR siga siendo 1, aún cuando volvamos a cambiar a 0 la entrada superior del biestable (Figura 1.4c). En resumen, la salida del biestable ha pasado a 1 y este valor de salida permanecerá incluso después de que la entrada superior vuelva a ser 0.

De forma similar, colocando temporalmente un valor 1 en la entrada inferior, forzaremos a que la salida del biestable sea 0 y este valor de salida permanecerá después de que el valor de entrada vuelva a 0.

Con la presentación del circuito biestable de las Figuras 1.3 y 1.4 pretendemos tres cosas distintas. En primer lugar, demostrar cómo pueden construirse dispositivos a partir de puertas lógicas, un proceso conocido con el nombre de diseño de circuitos digitales, que tiene una gran importancia en la ingeniería de computadoras. De hecho, el biestable es solo uno de los múltiples tipos de circuitos utilizados como herramientas básicas en la ingeniería de computadoras.

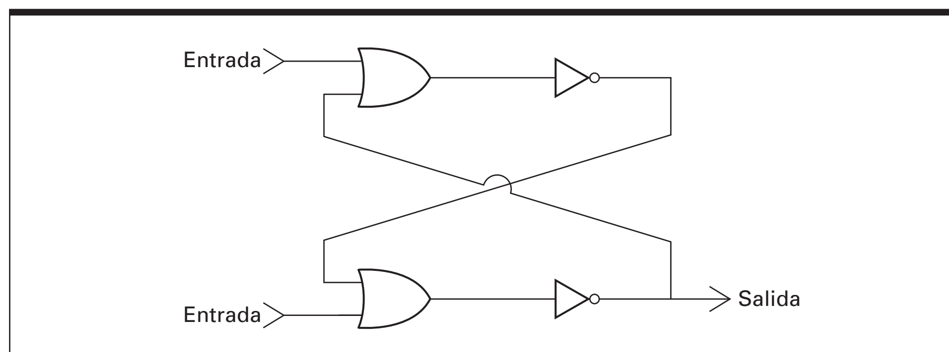
En segundo lugar, el concepto de biestable proporciona un ejemplo de abstracción y de uso de herramientas abstractas. De hecho, existen otras formas de construir un biestable. En la Figura 1.5 se muestra una alternativa. Si experimentamos con este circuito, podremos ver que, aunque tiene una estructura interna diferente, sus propiedades externas son las mismas que las del circuito de la Figura 1.3. Un ingeniero informático no necesita saber qué circuito se está empleando dentro de un biestable. En lugar de ello, solo necesita comprender las propiedades externas del biestable para poder utilizarlo como una herra-

Figura 1.4 Establecimiento de un valor 1 a la salida del biestable.



mienta abstracta. El biestable, junto con otros circuitos bien definidos, forma un conjunto de bloques componentes mediante los cuales un ingeniero puede construir circuitos más complejos. A su vez, el diseño de la circuitería de una computadora adopta una estructura jerárquica, en la que cada nivel utiliza como herramientas abstractas los componentes del nivel inferior.

Figura 1.5 Otra forma de construir un biestable.



El tercer objetivo de presentar el biestable es que este circuito constituye una de las formas de almacenar un bit dentro de una computadora moderna. Para ser más precisos, podemos hacer que un biestable tenga el valor de salida 0 o 1. Otros circuitos pueden ajustar este valor enviando pulsos a las entradas del biestable y otros circuitos adicionales pueden responder al valor almacenado utilizando la salida del biestable como entrada. Así, dentro de una computadora podemos emplear muchos biestables construidos como circuitos electrónicos de muy pequeño tamaño, con el propósito de grabar información que se codifica en forma de patrones de 0s y 1s. De hecho, la tecnología conocida como **VLSI** (*Very Large-Scale Integration*, Integración a muy gran escala), que permite construir millones de componentes electrónicos en una oblea (denominándose **chip** al circuito completo), se utiliza para crear dispositivos en miniatura que contienen millones de biestables, junto con su circuitería de control. A su vez, estos chips se utilizan como herramientas abstractas en la construcción de computadoras. De hecho, en algunos casos, se emplea la tecnología VLSI para crear una computadora completa en un único chip.

Notación hexadecimal

A la hora de considerar las actividades internas de una computadora, tenemos que tratar con patrones de bits, a los que nos referiremos con el nombre de cadenas de bits. Algunas de estas cadenas pueden tener una gran longitud. En ocasiones, a una cadena larga de bits se le denomina **flujo de bits**. Lamentablemente, es difícil para la mente humana imaginar y comprender los flujos de bits. La simple transcripción del patrón de bits 101101010011 resulta tediosa y puede provocar errores de transcripción. Para simplificar la representación de dichos patrones de bits, solemos utilizar por tanto una notación abreviada denominada **notación hexadecimal**, que aprovecha el hecho de que los patro-

Figura 1.6 Sistema de codificación hexadecimal.

Patrón de bits	Representación hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

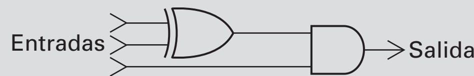
nes de bits dentro de una máquina tienden a tener longitudes que son múltiplos de cuatro. En particular, la notación hexadecimal utiliza un único símbolo para representar un patrón de cuatro bits. Por ejemplo, una cadena de doce bits puede representarse mediante tres símbolos hexadecimales.

En la Figura 1.6 se presenta el sistema de codificación hexadecimal. La columna de la izquierda muestra todos los posibles patrones de bits de longitud igual a cuatro; la columna de la derecha muestra el símbolo utilizado en notación hexadecimal para representar el patrón de bits situado a su izquierda. Utilizando este sistema, el patrón de bits 10110101 se representa mediante los dígitos hexadecimales B5. Esto se obtiene dividiendo el patrón de bits en subcadenas de longitud igual a cuatro y luego representando cada subcadena mediante su equivalente hexadecimal, 1011 se representa mediante B y 0101 se representa mediante 5. De esta forma, el patrón de 16 bits 1010010011001000 puede reducirse a la forma hexadecimal A4C8, que es mucho más manejable.

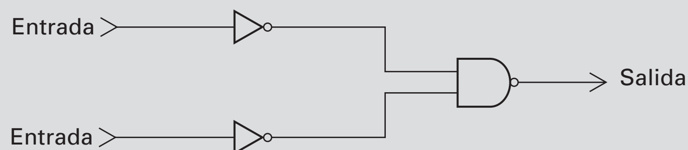
En el siguiente capítulo utilizaremos ampliamente la notación hexadecimal y tendremos la oportunidad de apreciar lo eficiente que es dicha notación.

Cuestiones y ejercicios

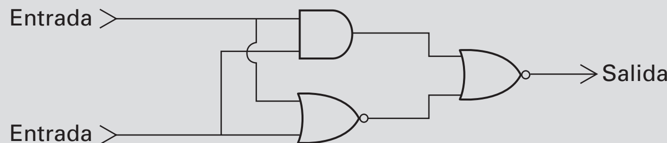
1. ¿Qué patrones de bits de entrada harán que el siguiente circuito genere una salida igual a 1?



2. En el texto, afirmamos que colocar un 1 en la entrada inferior del biestable de la Figura 1.3 (mientras que se mantiene la entrada superior a 0) fuerza a la salida del biestable a tomar el valor 0. Describa la secuencia de sucesos que tienen lugar dentro del biestable en este caso.
3. Suponiendo que ambas entradas del biestable de la Figura 1.5 están a 0, describa la secuencia de sucesos que tienen lugar cuando fijamos temporalmente un valor igual a 1 en la entrada superior.
4. **a.** Si hacemos pasar la salida de una puerta AND a través de una puerta NOT, la combinación de ambas puertas calcula la operación booleana NAND, que proporciona una salida igual a 0 únicamente cuando sus dos entradas están a 1. El símbolo para una puerta NAND es igual que el de la puerta AND, salvo porque tiene un círculo a la salida. El siguiente circuito contiene una puerta NAND. ¿Qué operación booleana realiza este circuito?



b. Si se aplica la salida de una puerta OR a una puerta NOT, la combinación de ambas puertas calcula la operación booleana denominada NOR, cuya salida es igual a 1 únicamente cuando sus dos entradas son iguales 0. El símbolo de una puerta NOR es el mismo que el de una puerta OR salvo porque incluye un círculo a la salida. El siguiente circuito está formado por una puerta AND y dos puertas NOR. ¿Qué operación booleana realiza este circuito?



5. Utilice la notación hexadecimal para representar los siguientes patrones de bits:

a. 0110101011110010 **b.** 111010000101010100010111
c. 01001000

6. ¿Qué patrones de bits están representados por los siguiente patrones de dígitos hexadecimales?

a. 5FD97 **b.** 610A **c.** ABCD **d.** 0100

1.2 Memoria principal

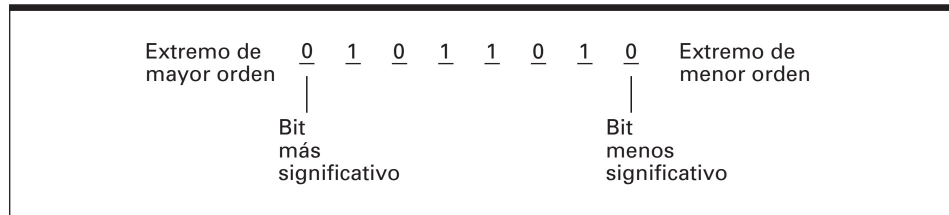
Con el objetivo de almacenar datos, una computadora contiene un enorme conjunto de circuitos (tales como biestables), cada uno de los cuales es capaz de almacenar un único bit. Este conjunto de bits se conoce como **memoria principal** de la máquina.

Organización de la memoria

La memoria principal de una computadora está organizada en una serie de unidades accesibles denominadas **celdas**, siendo el tamaño típico de celda igual a ocho bits. (Una cadena de ocho bits se denomina **byte**. Por tanto, una celda de memoria típica tiene una capacidad de un byte.) Las computadoras pequeñas utilizadas en electrodomésticos tales como los hornos microondas pueden disponer de memorias principales compuestas por solo unos pocos cientos de celdas, mientras que las computadoras de mayor tamaño pueden tener miles de millones de celdas en su memoria principal.

Aunque no existen los conceptos de izquierda ni derecha en una computadora, normalmente tendemos a conceptualizar los bits de una celda de memoria como si estuvieran organizados en una fila. El extremo izquierdo de esta fila se denomina **extremo de mayor peso** y el extremo derecho se denomina **extremo de menor peso**. El bit más a la izquierda se denomina bit de mayor peso o **bit más significativo** para hacer referencia al hecho de que si interpretáramos el

Figura 1.7 Organización de una celda de memoria de un byte.

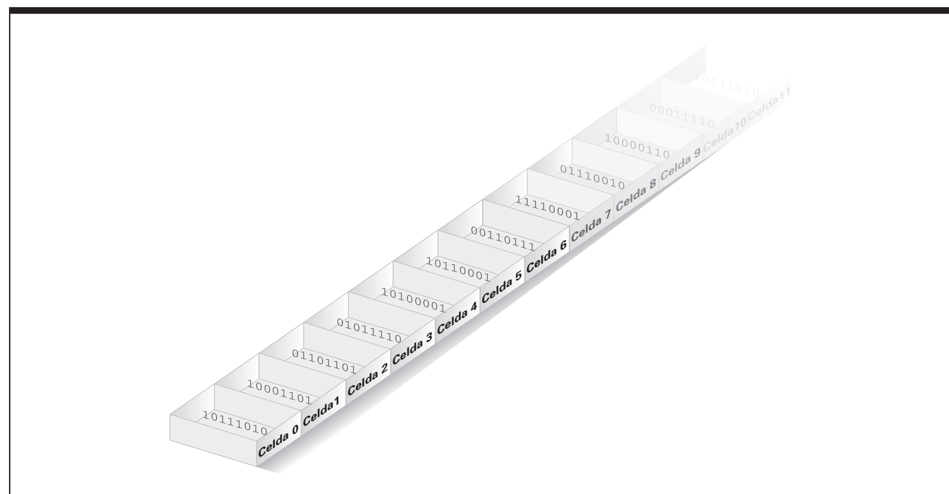


contenido de la celda como si representaran un valor numérico, dicho bit sería el dígito más significativo del número. De forma similar, el bit situado más a la derecha es el bit de menor peso o **bit menos significativo**. Por tanto, podemos representar el contenido de una celda de memoria de tamaño igual a un byte como se muestra en la Figura 1.7.

Para identificar a cada una de las celdas individuales que forman la memoria principal de una computadora, a cada celda se le asigna un "nombre" distintivo, denominado **dirección**. El sistema es análogo a la técnica de identificar las viviendas dentro de una ciudad mediante su dirección postal. Sin embargo, en el caso de las celdas de memoria, las direcciones utilizadas son completamente numéricas. Para ser más precisos, es como si todas las celdas estuvieran colocadas en una única fila y fueran numeradas por orden comenzando por el valor cero. Dicho sistema de direccionamiento no solo nos proporciona una forma de identificar a cada celda de forma unívoca, sino que también asigna un orden a las propias celdas (Figura 1.8), permitiéndonos emplear expresiones tales como "la celda siguiente" o "la celda anterior".

Una consecuencia importante de asignar un orden tanto a las celdas de la memoria principal como a los bits que componen cada celda es que la colección completa de bits que componen la memoria principal de la computadora está, en esencia, ordenada como si formara una única fila de gran longitud. Por esto, podemos utilizar fragmentos de esa fila para almacenar patrones de bits que puedan tener una longitud mayor que la de una sola celda. Por ejemplo,

Figura 1.8 Celdas de memoria ordenadas según su dirección.



podemos almacenar una cadena de 16 bits simplemente utilizando dos celdas de memoria consecutivas.

Para completar la memoria principal de una computadora, combinamos la circuitería que almacena los bits con la circuitería necesaria para que otros circuitos puedan almacenar y extraer datos de las celdas de memoria. De esta forma, otros circuitos pueden extraer datos de la memoria preguntando electrónicamente cuál es el contenido de una cierta dirección (lo que se denomina operación de lectura) o pueden almacenar información en la memoria solicitando que se coloque un determinado patrón de bits en la celda de memoria situada en una dirección concreta (lo que se conoce como operación de escritura).

Puesto que la memoria principal de una computadora está organizada en forma de celdas individuales direccionables, se puede acceder de manera independiente al contenido de cada celda, según sea necesario. Para reflejar esa capacidad de acceder a las celdas en cualquier orden, a la memoria principal de la computadora se la denomina a menudo **memoria de acceso aleatorio** (RAM, *Random Access Memory*). Esta característica de acceso aleatorio a la memoria principal contrasta enormemente con los sistemas de almacenamiento masivo que veremos en la siguiente sección, en los que las cadenas de bits de gran longitud se manipulan en forma de bloques completos.

Aunque hemos presentado los biestables como un método para el almacenamiento de bits, la RAM de la mayor parte de las computadoras modernas se construye empleando otras tecnologías, que proporcionan una mayor miniaturización y un tiempo de respuesta más rápido. Muchas de estas tecnologías almacenan los bits en forma de pequeñas cargas eléctricas que se disipan rápidamente. En consecuencia, dichos dispositivos requieren circuitería adicional, conocida como circuito de refresco, que restaura repetidamente las cargas muchas veces por segundo. Para dejar clara esta volatilidad, a la memoria de una computadora construida con ese tipo de tecnología se la denomina a menudo **memoria dinámica**, lo que ha dado lugar al término **DRAM** (*Dynamic* RAM, RAM dinámica). En ocasiones también se emplea el término **SDRAM** (*Synchronous* DRAM, DRAM síncrona), para referirse a la memoria DRAM en la que se aplican técnicas adicionales con el fin de reducir el tiempo necesario para extraer el contenido de sus celdas de memoria.

Medida de la capacidad de memoria

Como veremos en el siguiente capítulo, es conveniente diseñar los sistemas de memoria principal de forma que el número total de celdas sea una potencia de dos. A su vez, el tamaño de las memorias en las antiguas computadoras se medía a menudo en múltiplos de 1024 (2^{10}) celdas. Puesto que 1024 es un valor muy próximo a 1000, la comunidad informática ha adoptado el prefijo *kilo* para hacer referencia a esta unidad. Es decir, el término *kilobyte* (abreviado como KB) se utilizaba para referirse a 1024 bytes. Por tanto, de una máquina con 4096 celdas de memoria se decía que tenía una memoria de 4KB ($4096 = 4 \times 1024$). A medida que las memorias fueron teniendo mayor tamaño, esta terminología se amplió para incluir el MB (megabyte), el GB (gigabyte) y el TB (terabyte). Lamentablemente, esta aplicación de los prefijos *kilo-*, *mega-*, etc., representa una utilización incorrecta de la terminología, puesto que esos prefijos se emplean en otros campos para hacer referencia a unidades que son potencias de mil. Por ejemplo, a la hora de medir

distancias, la palabra *kilómetro* representa 1000 metros y cuando se miden frecuencias de radio, un *megahercio* representa 1.000.000 de hercios. Por tanto, conviene tener cuidado al emplear esta terminología. Como regla general, los prefijos como *kilo-*, *mega-*, etc. hacen referencia a potencias de dos cuando se les utiliza en el contexto de la memoria de una computadora, mientras que en otros contextos hacen referencia a potencias de mil.

Cuestiones y ejercicios

1. Si la celda de memoria cuya dirección es 5 contiene el valor 8, ¿cuál será la diferencia entre escribir el valor 5 en la celda número 6 y desplazar el contenido de la celda número 5 a la celda número 6?
2. Suponga que desea intercambiar los valores almacenados en las celdas de memoria 2 y 3. ¿Cuál es el error en la siguiente secuencia de pasos?

   Paso 1. Desplazar el contenido de la celda número 2 a la celda número 3.

   Paso 2. Desplazar el contenido de la celda número 3 a la celda número 2.

   Diseñe una secuencia de pasos que permita intercambiar correctamente el contenido de estas celdas. En caso necesario, puede utilizar celdas adicionales.
3. ¿Cuántos bits habrá en la memoria de una computadora de 4KB?

1.3 Almacenamiento masivo

Debido a la volatilidad y al tamaño limitado de la memoria principal de una computadora, la mayoría de las computadoras disponen de dispositivos de almacenamiento adicionales conocidos como sistemas de **almacenamiento masivo** (o de almacenamiento secundario), entre los que se incluyen los discos magnéticos, los discos CD y DVD, las cintas magnéticas y las unidades flash, de los que hablaremos más adelante. Entre las ventajas de los sistemas de almacenamiento masivo, con respecto a la memoria principal, podemos citar una menor volatilidad, mayores capacidades de almacenamiento, su bajo coste y, en muchos casos, la capacidad de extraer el medio de almacenamiento de la máquina, con el propósito de archivarlo.

A menudo se emplean los términos *en línea* y *fuera de línea* para describir a aquellos dispositivos que pueden estar conectados o no a una máquina. Estar **en línea** significa que el dispositivo o la información están conectados y la máquina puede acceder fácilmente a ellos, sin que haya intervención humana. Estar **fuera de línea** significa que se requiere intervención humana para que la máquina pueda acceder al dispositivo o la información, quizá porque el dispositivo debe encenderse o porque el medio en el que se almacena la información debe introducirse en algún tipo de mecanismo.

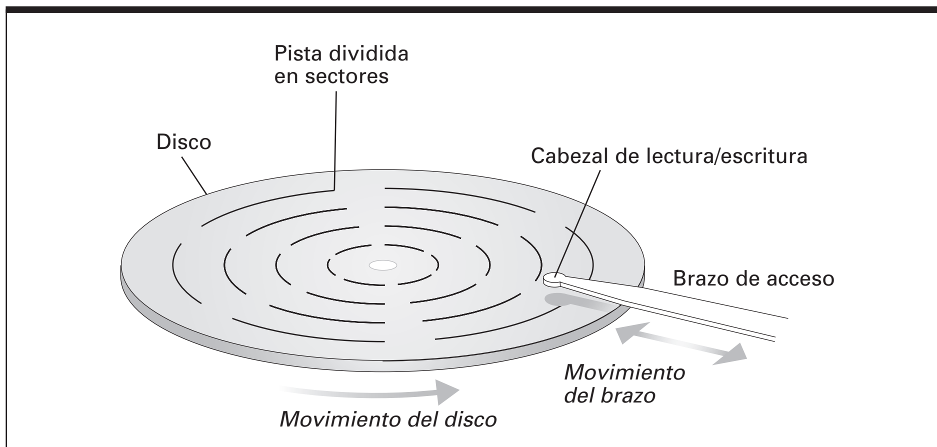
Una de las principales desventajas de los sistemas de almacenamiento masivo es que normalmente requieren algún tipo de movimiento mecánico, por lo que hace falta un tiempo considerablemente mayor para almacenar y extraer los datos del que se necesita en la memoria principal de la computadora, en la que todas las actividades se llevan a cabo de forma electrónica.

Sistemas magnéticos

Durante años, la tecnología magnética ha sido la dominante dentro del sector del almacenamiento masivo. El ejemplo más común y que todavía se utiliza es el **disco magnético**, en el que se utiliza un fino disco giratorio con un recubrimiento magnético para almacenar los datos (Figura 1.9). Los cabezales de lectura/escritura se colocan por encima y/o por debajo del disco, de modo que cuando el disco gira, cada cabezal recorre un círculo, denominado **pista**. Reposicionando los cabezales de lectura/escritura puede accederse a las distintas pistas concéntricas. En muchos casos, un sistema de almacenamiento en disco está formado por varios discos montados sobre un eje común, apilados unos encima de otros, dejando el espacio suficiente para poder deslizar los cabezales de lectura/escritura entre un disco y otro. En tales casos, los cabezales de lectura/escritura se mueven al unísono. Cada vez que se reposicionan los cabezales de lectura/escritura, pasa a estar accesible un nuevo conjunto de pistas, que se denomina **cilindro**.

Puesto que una pista puede contener más información de la que normalmente deseamos manipular en cualquier momento determinado, cada pista se divide en pequeños arcos denominados **sectores** en los que la información se almacena en forma de una cadena continua de bits. Todos los sectores de un disco contienen el mismo número de bits (capacidades típicas van desde 512 bytes a unos pocos KB), y en los sistemas de almacenamiento en disco más simples cada pista contiene un mismo número de sectores. Por tanto, los bits contenidos en un sector de un pista que esté cerca del borde exterior del disco estarán almacenados de forma menos compacta que aquellos que se encuentren en las pistas situadas más cerca del centro, debido a que las pistas externas son más largas que las internas. De hecho, en los sistemas de almacenamiento en disco de alta capacidad, las pistas ubicadas cerca del borde exterior son capaces de almacenar un número significativamente mayor de sectores que las situadas cerca del centro, y esta capacidad se utiliza a menudo aplicando una técnica denominada **grabación de bits por zonas**. Utilizando esta técnica, se divide el disco en una serie de zonas compuestas por varias pistas adyacentes, estando un disco típico dividido en aproximadamente diez zonas. Todas las pis-

Figura 1.9 Sistema de almacenamiento en disco.



tas de una zona tienen el mismo número de sectores, pero cada zona tiene más sectores por pista que las zonas contenidas dentro de ella. De esta manera, se consigue una utilización más eficiente de la superficie completa del disco. Independientemente de los detalles, cualquier sistema de almacenamiento en disco está compuesto de muchos sectores individuales, a cada uno de los cuales se puede acceder en forma de una cadena independiente de bits.

La ubicación de las pistas y de los sectores no es una característica permanente de la estructura física de un disco. En lugar de ello, esas pistas y sectores se marcan de forma magnética a través de un proceso conocido como **formateo** (o inicialización) del disco. Este proceso suele ser realizado por el fabricante del disco, con lo que se comercializan lo que denominamos discos formateados. La mayoría de las computadoras también pueden llevar a cabo esta tarea. Así, si resultara dañada la información de formato en un disco, se puede reformatear, aunque dicho proceso destruye toda la información que hubiera sido grabada anteriormente en el disco.

La capacidad de un sistema de almacenamiento en disco depende del número de discos apilados que se utilizan y de la densidad con la que se coloquen las pistas y sectores. Los sistemas de menor capacidad pueden estar formados por un único disco. Los sistemas de disco de alta capacidad, capaces de almacenar muchos gigabytes o incluso terabytes de datos, pueden estar compuestos por entre tres y seis discos montados sobre un eje común. Además, los datos pueden almacenarse tanto en la superficie superior como en la inferior de cada disco.

Se utilizan diversas medidas para evaluar el rendimiento de un sistema de disco: (1) **tiempo de búsqueda** (el tiempo requerido para desplazar los cabezales de lectura/escritura de una pista a otra); (2) **retardo de rotación** o **tiempo de latencia** (la mitad del tiempo requerido para que el disco realice una rotación completa, lo cual es igual al tiempo medio requerido para que los datos deseados giren hasta colocarse bajo el cabezal de lectura/escritura una vez que ese cabezal ha sido posicionado sobre la pista deseada); (3) **tiempo de acceso** (la suma del tiempo de búsqueda y del retardo de rotación) y (4) **tasa de transferencia** (la tasa a la que pueden transferirse datos hacia o desde el disco). Observe que en el caso de la técnica de grabación de bits por zonas, la cantidad de datos que pasan bajo un cabezal de lectura/escritura en una única rotación del disco es mayor para las pistas situadas en una zona externa que para las que están en una zona más interna, por lo que la tasa de transferencia de datos varía dependiendo de la zona del disco que se esté empleando.

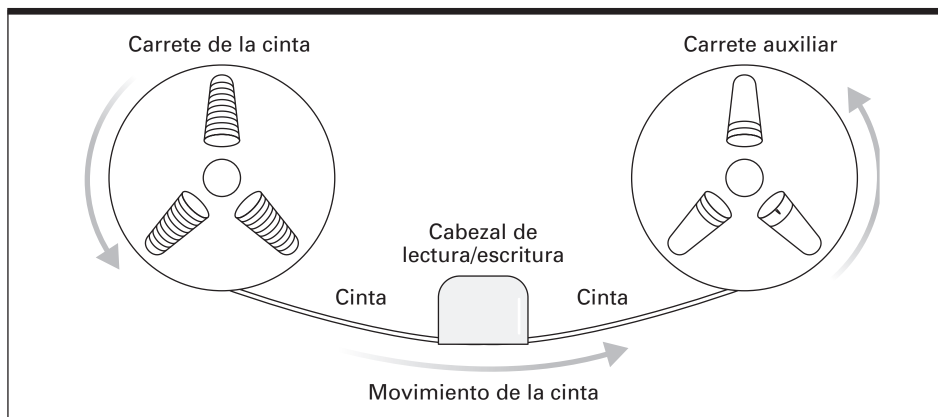
Un factor que limita el tiempo de acceso y la tasa de transferencia es la velocidad a la que gira el sistema de disco. Para conseguir velocidades de rotación altas, los cabezales de lectura/escritura de dichos sistemas no tocan el disco, sino que "flotan" a corta distancia de la superficie. La separación es tan pequeña que incluso una minúscula partícula de polvo podría quedar atascada entre el cabezal y la superficie del disco, destruyendo ambos (un fenómeno conocido con el nombre de choque de cabezal). Por ello, los sistemas de disco suelen montarse en carcasas herméticas, que se sellan en la fábrica. Con este tipo de montaje, los sistemas de disco son capaces de girar a velocidades de varios miles de rotaciones por segundo, consiguiéndose tasas de transferencia del orden de MB por segundo.

Puesto que los sistemas de disco requieren un movimiento físico para poder operar, las velocidades que permiten alcanzar son inferiores a las de los circuitos electrónicos. Los retardos dentro de un circuito electrónico se miden en nanosegundos (mil millonésima parte de un segundo) o menos, mientras que los tiempos de búsqueda, de latencia y de acceso de disco se miden en milisegundos (milésimas de segundo). Por tanto, el tiempo requerido para recuperar información de un sistema de disco puede parecerle una eternidad al circuito electrónico que está esperando el resultado.

Los sistemas de almacenamiento en disco no son el único tipo de dispositivo de almacenamiento masivo que emplea tecnología magnética. Un tipo más antiguo de almacenamiento masivo en el se utiliza tecnología magnética es la **cinta magnética** (Figura 1.10). En estos sistemas, la información se almacena en el recubrimiento magnético de una cinta delgada de plástico que está devanada sobre un carrete para poder almacenarla. Cuando se quiere acceder a los datos, se monta la cinta en un dispositivo denominado unidad de cinta, que normalmente puede leer, escribir y rebobinar la cinta bajo control de la computadora. Las unidades de cinta tienen tamaños muy variables, que van desde las pequeñas unidades de cartucho, que utilizan cintas similares en apariencia a las de los equipos de sonido comerciales, hasta las unidades más antiguas que emplean dos carretes de cinta. Aunque la capacidad de estos dispositivos depende del formato utilizado, la mayoría puede almacenar varios GB.

Una de las principales desventajas de la cinta magnética es que al desplazarse entre diferentes posiciones de la cinta puede consumir muchísimo tiempo, debido a la cantidad tan grande de cinta que hay que desplazar entre los carretes. Debido a ello, los sistemas de cinta tienen tiempos de acceso a los datos mucho mayores que los sistemas de disco magnético, en los que se puede acceder a los diferentes sectores mediante cortos movimientos del cabezal de lectura/escritura. Por ello, los sistemas de cinta no son muy populares para el almacenamiento de datos en línea. En lugar de ello, la tecnología de cinta magnética se reserva para las aplicaciones de almacenamiento de datos en archivos fuera de línea donde su alta capacidad, su fiabilidad y su bajo coste resulten convenientes, aunque los avances en otras tecnologías alternativas, como los DVD y las unidades flash están poniendo en serios aprietos a estos últimos vestigios de la tecnología de cinta magnética.

Figura 1.10 Mecanismo de almacenamiento en cinta magnética.



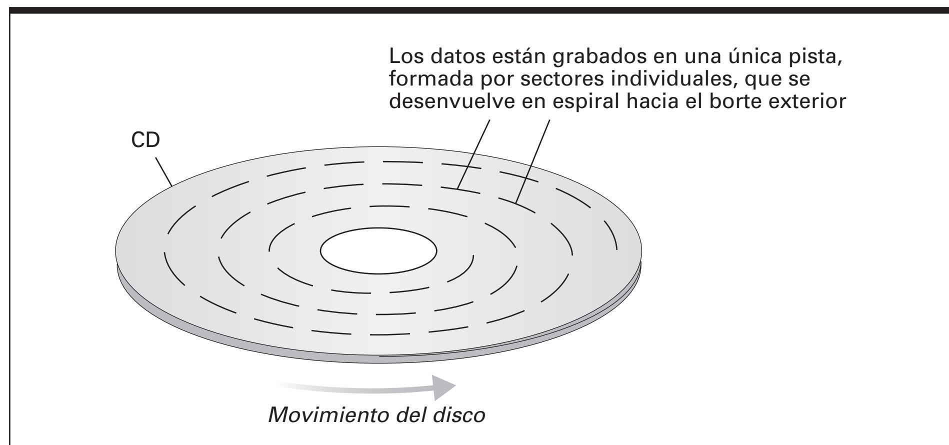
Sistemas ópticos

Otra clase de sistemas de almacenamiento masivo utiliza la tecnología óptica. Un ejemplo es el **disco compacto** (CD, *Compact Disk*). Estos discos tienen 12 centímetros de diámetro y están compuestos de un material reflectante, cubierto con una capa protectora transparente. La información se almacena en estos discos creando variaciones en sus superficies reflectantes. Esta información se puede entonces extraer mediante un rayo láser que detecta las irregularidades en la superficie reflectante del CD, a medida que este gira.

La tecnología de CD se aplicó originalmente a las grabaciones de sonido, utilizando un formato de grabación conocido como **CD-DA** (*Compact Disk-Digital Audio*, Disco compacto-sonido digital) y los CD utilizados hoy día para el almacenamiento de datos emplean esencialmente el mismo formato. En particular, la información en estos CD se almacena en una única pista que gira en espiral alrededor del CD, como el surco de uno de esos discos de vinilo ya pasados de moda. Sin embargo, a diferencia de esos discos de vinilo, la pista de un CD se desarrolla en espiral desde el interior hacia el exterior (Figura 1.11). Esta pista está dividida en unidades denominadas sectores, cada uno con sus propias marcas de identificación y con una capacidad de 2KB de datos, que equivale a $^{1}/_{75}$ de un segundo de música en el caso de las grabaciones de audio.

Observe que la distancia recorrida por la pista en espiral es mayor en las proximidades del borde externo del disco que en las partes situadas más cerca del centro. Para maximizar la capacidad de un CD, la información se almacena con una densidad lineal uniforme a lo largo de toda la pista en espiral, lo que quiere decir que hay más información almacenada en uno de los círculos externos de la espiral que en uno de los círculos internos. A su vez, se leerán más sectores en una única revolución del disco cuando el rayo láser esté explorando la parte externa de la pista en espiral que cuando esté explorando la parte interna de la misma. Por tanto, para obtener una tasa de transferencia de datos uniforme, los reproductores de CD-DA están diseñados para que la velocidad de rotación varíe dependiendo de la posición del rayo láser. Sin embargo, la mayoría de los sistemas de CD empleados para el almacenamiento de datos en una computadora giran a una velocidad más rápida que los CD de audio, y que

Figura 1.11 Formato de almacenamiento en CD.



además es constante, por lo que necesitan poder tolerar las variaciones en las transferencias de las tasas de datos.

Como consecuencia de estas decisiones de diseño, los sistemas de almacenamiento en CD presentan un mejor rendimiento cuando se manejan largas cadenas continuas de datos, que es lo que sucede por ejemplo al reproducir música. Por el contrario, cuando una aplicación requiere acceder a elementos de datos de una forma aleatoria, la técnica empleada para el almacenamiento en discos magnéticos (pistas individuales y concéntricas divididas en sectores accesibles de manera individual) es mucho mejor que la pista en espiral utilizada en los CD.

Los CD tradicionales tienen capacidades que van desde los 600 a los 700 MB. Sin embargo, los **DVD** (*Digital Versatile Disks*, discos digitales versátiles), que están construidos a partir de múltiples capas semitransparentes que actúan como superficies distintas cuando se las ilumina mediante un láser con un enfoque muy preciso, proporcionan capacidades de almacenamiento de varios GB. Dichos discos son capaces de almacenar largas presentaciones multimedia, incluyendo películas comerciales completas. Finalmente, la tecnología Blu-ray, que utiliza un láser en el espectro azul-violeta de la luz (en lugar de un láser rojo) es capaz de enfocar el rayo láser con una precisión muy grande. Como resultado, los **BD** (*Blu-ray Disks*, discos blu-ray) proporcionan una capacidad cinco veces superior a la de un DVD. Esta capacidad de almacenamiento tan grande es necesaria para poder satisfacer las demandas del mercado del vídeo de alta definición.

Unidades flash

Una propiedad común de los sistemas de almacenamiento masivo basados en tecnología magnética u óptica es que hace falta un cierto movimiento físico para almacenar y extraer los datos; por ejemplo, son necesarios discos giratorios, cabezales de lectura/escritura móviles y rayos láser que se enfocan en un punto u otro. Esto significa que el almacenamiento y la extracción de datos son lentos comparados con la velocidad de los circuitos electrónicos. La tecnología de las **memorias flash** tiene el potencial de acabar con esta desventaja. En un sistema de memoria flash, los bits se almacenan enviando señales electrónicas al medio de almacenamiento en el que hacen que los electrones queden atrapados en pequeñas cámaras de dióxido de silicio, alterando de esa manera una serie de pequeños circuitos electrónicos. Puesto que estas cámaras son capaces de mantener cautivos esos electrones durante muchos años, esta tecnología resulta adecuada para el almacenamiento de datos fuera de línea.

Aunque se puede acceder a los datos almacenados en sistemas de memoria flash en unidades compuestas por un número pequeño de bytes, como en las aplicaciones RAM, la tecnología actual requiere que los datos almacenados se borren en bloques de gran tamaño. Además, los borrados sucesivos van dañando lentamente las cámaras de dióxido de silicio, lo que quiere decir que la tecnología de memoria flash actual no es adecuada para las aplicaciones generales de una memoria principal, en la que los contenidos de la memoria pueden verse alterados muchas veces por segundo. Sin embargo, en aquellas aplicaciones en las que las modificaciones puedan controlarse hasta un nivel razonable, como sucede por ejemplo en las cámaras digitales, los teléfonos celulares y los PDA portátiles, la memoria flash se ha convertido en la tecno-

logía de almacenamiento preferida. De hecho, puesto que la memoria flash no es sensible a los golpes (a diferencia de lo que sucede con los sistemas magnéticos y ópticos), su potencial en las aplicaciones de tipo portátil es enorme.

Para aplicaciones de carácter masivo de carácter general, hay disponibles dispositivos de memoria flash denominados **unidades flash**, con capacidades de hasta unos cuantos centenares de gigabytes. Estas unidades se integran en pequeñas carcasas de plástico, de aproximadamente unos siete centímetros de longitud, con una tapa extraíble en uno de los extremos, para proteger el conector eléctrico cuando la unidad está fuera de línea. La alta capacidad de estas unidades portátiles, así como el hecho de que puedan conectarse y desconectarse fácilmente a una computadora las hacen ideales para el almacenamiento de datos fuera de línea. Sin embargo, la vulnerabilidad de sus diminutas cámaras de almacenamiento indica que no son tan fiables como los discos ópticos para aplicaciones de almacenamiento a muy largo plazo.

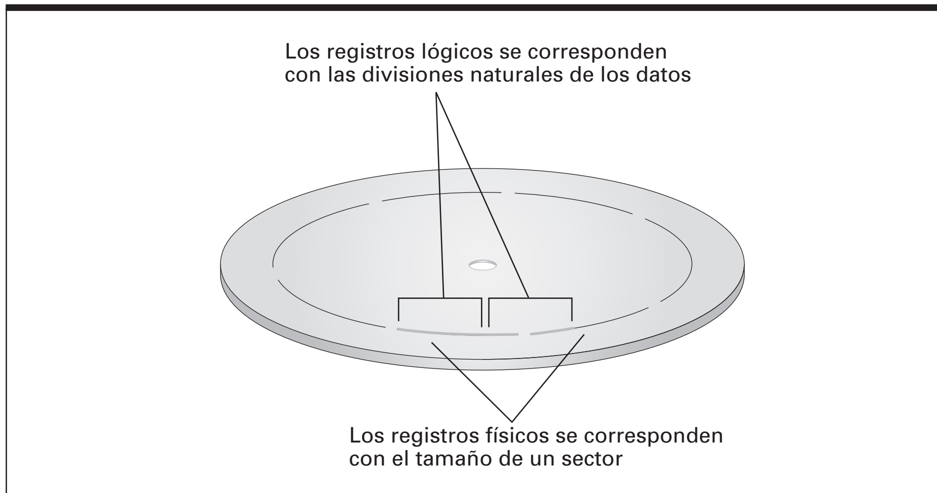
Otra aplicación de la tecnología flash es la que podemos encontrar en las **tarjetas de memoria SD** (*Secure Digital*), que a veces se denominan ***SD Card***. Estas tarjetas proporcionan hasta dos GB de almacenamiento y se encapsulan en una oblea recubierta de plástico del tamaño aproximado de un sello de correos (las tarjetas SD también están disponibles en tamaños mini y micro más pequeños), las **tarjetas de memoria SDHC** (*High Capacity*, alta capacidad) pueden proporcionar hasta 32 GB y la siguiente generación de **tarjetas de memoria SDXC** (*Extended Capacity*, capacidad ampliada) pueden tener más de un TB. Dado su pequeño tamaño, estas tarjetas pueden insertarse cómodamente en las ranuras de pequeños dispositivos electrónicos. Por tanto, son ideales para cámaras digitales, teléfonos inteligentes, reproductores de música, sistemas de navegación para vehículos y muchos otros tipos de dispositivos electrónicos.

Almacenamiento y extracción de archivos

La información almacenada en un sistema de almacenamiento masivo está agrupada de manera conceptual en unidades de gran tamaño denominadas **archivos**. Un archivo típico puede contener un documento de texto completo, una fotografía, un programa, una grabación de música o una colección de datos acerca de los empleados de una empresa. Hemos visto que los dispositivos de almacenamiento masivo exigen que estos archivos se almacenen y extraigan en unidades más pequeñas, formadas por varios bytes. Por ejemplo, un archivo almacenado en un disco magnético debe manipularse por sectores, cada uno de los cuales es de un tamaño fijo predeterminado. Un bloque de datos que esté adaptado a las características específicas de un dispositivo de almacenamiento se denomina **registro físico**. Así, un archivo de gran tamaño en un dispositivo de almacenamiento masivo estará normalmente formado por muchos registros físicos.

Por contraste con esta división en registros físicos, los archivos suelen tener divisiones naturales que están determinadas por la información representada. Por ejemplo, un archivo que contenga información concerniente a los empleados de una empresa estará compuesto de múltiples unidades, cada una de las cuales constará de la información relativa a un empleado concreto. O bien, un archivo que contenga un documento de texto constará de párrafos o páginas. Estos bloques de datos que aparecen de manera natural se denominan **registros lógicos**.

Figura 1.12 Registros lógicos y registros físicos en un disco.



Los registros lógicos están formados a menudo por unidades más pequeñas denominados **campos**. Por ejemplo, un registro lógico que contenga información acerca de un empleado estará compuesto probablemente por campos tales como el nombre, la dirección, el número de identificación del empleado, etc. En ocasiones, cada registro lógico de un archivo se identifica de manera unívoca por medio de un campo concreto del registro (por ejemplo, un número de identificación de empleado, un código de componente o un número de elemento de un catálogo). Dicho campo de identificación se denomina **campo clave**. El valor almacenado en el campo clave se denomina **clave**.

Los tamaños de los registros lógicos no suelen corresponderse con el tamaño del registro físico impuesto por un dispositivo de almacenamiento masivo. Debido a ello, podemos encontrarnos con que haya varios registros lógicos dentro de un mismo registro físico, o también con que un registro lógico esté repartido entre dos o más registros físicos (Figura 1.12). El resultado es que hace falta una cierta tarea de manipulación a la hora de extraer datos de los sistemas de almacenamiento masivo. Una solución común a este problema consiste en reservar un área de la memoria principal lo suficientemente grande como para almacenar varios registros físicos y emplear dicho espacio de memoria como área de reordenación. Es decir, podemos transferir bloques de datos compatibles con el tamaño de los registros físicos entre esta área de la memoria principal y el sistema de almacenamiento masivo, mientras que los datos que residen en esa área de la memoria principal pueden referenciarse en términos de los registros lógicos.

Un área de memoria utilizada de este modo se denomina **buffer**. En general, un buffer es un área de almacenamiento que se emplea para albergar datos de manera temporal, normalmente durante el proceso de transferirlos de un dispositivo a otro. Por ejemplo, las impresoras modernas contienen su propia circutería de memoria y gran parte de ella se utiliza como buffer para almacenar partes de un documento que han sido recibidas por la impresora pero aún no se han impreso.

Cuestiones y ejercicios

1. ¿Qué ventaja obtenemos incrementando la velocidad de rotación de un disco o CD?
2. A la hora de grabar datos en un sistema de almacenamiento de múltiples discos, ¿debemos rellenar completamente la superficie de un disco antes de comenzar en otra superficie, o debemos rellenar un cilindro completo antes de empezar con el siguiente?
3. ¿Por qué deberíamos almacenar los datos en un sistema de reservas de vuelos que está siendo actualizado constantemente en un disco magnético y no en un CD o DVD?
4. En ocasiones, al modificar un documento con un procesador de textos, el añadir más texto no incrementa el tamaño aparente del archivo en el dispositivo de almacenamiento masivo, mientras que en otras ocasiones la adición de un solo símbolo puede incrementar el tamaño aparente del archivo en varios cientos de bytes. ¿Por qué?
5. ¿Qué ventajas tienen las unidades flash con respecto a otros sistemas de almacenamiento masivo presentados en esta sección?
6. ¿Qué es un buffer?

1.4 Representación de la información mediante patrones de bits

Habiendo visto las técnicas para el almacenamiento de bits, vamos a ver ahora cómo puede representarse la información mediante patrones de bits. Nuestro estudio se va a centrar en los métodos más populares para codificar texto, datos numéricos, imágenes y sonido. Cada uno de estos sistemas tiene repercusiones que suelen ser visibles para el usuario típico de computadoras. Nuestro objetivo es comprender estas técnicas lo suficiente como para poder reconocer sus consecuencias cuando se presenten.

Representación de textos

La información en forma de texto se suele representar por medio de un código en el que se asigna un patrón determinado de bits a cada uno de los distintos símbolos que aparecen en el texto (como por ejemplo las letras del alfabeto y los signos de puntuación). El texto se representa entonces mediante una larga cadena de bits, en la que los sucesivos patrones representan los símbolos sucesivos del texto original.

En la década de 1940 y 1950, se diseñaron y utilizaron muchos de esos códigos con diferentes tipos de equipos, lo que generó una lógica proliferación de problemas de comunicación. Para aliviar esta situación, el instituto **ANSI** (**American National Standards Institute**, Instituto Nacional Estadounidense de Estandarización) adoptó el código **ASCII** (**American Standard Code for Information Interchange**, Código estándar americano para el intercambio de

información). Este código utiliza patrones de siete bits para representar las letras mayúsculas y minúsculas del alfabeto inglés, además de los signos de puntuación, los dígitos 0 a 9 y cierta información de control, como por ejemplo las indicaciones de avance de línea, retorno de carro y tabulación. ASCII puede ampliarse a un formato de ocho bits por símbolo, añadiendo un 0 en el extremo más significativo de cada uno de los patrones de siete bits. Esta técnica no solo permite obtener un código en el que cada patrón encaja convenientemente en las celdas típicas de memoria, con un tamaño igual a un byte, sino que también proporciona 128 patrones de bits adicionales (los que se obtienen asignando al bit extra el valor 1), dichos patrones adicionales pueden utilizarse para representar símbolos no contenidos en el alfabeto inglés y en su conjunto de signos de puntuación asociado.

En el Apéndice A se presenta una parte del código ASCII en su formato de ocho bits por símbolo. Consultando dicho apéndice, podemos ver que el patrón de bits

```
01001000  01100101  01101100  01101100  01101111  00101110
```

representa el mensaje “Hello.” , como se ilustra en la Figura 1.13.

La **Organización Internacional de Estandarización (International Organization for Standardization)**, también denominada **ISO** (en referencia a la palabra griega *isos*, que significa igual) ha desarrollado un serie de extensiones del código ASCII, cada una de las cuales se diseñó para satisfacer las necesidades de cada uno de los grupos de lenguajes existentes. Por ejemplo, uno de los estándares proporciona los símbolos necesarios para escribir texto en la mayoría de los idiomas hablados en Europa occidental. Entre sus 128 patrones adicionales se encuentran los símbolos correspondientes a la libra inglesa, a las vocales alemanas ä, ö y ü y a las vocales acentuadas del español.

Los estándares ASCII ampliados definidos por ISO representaron un gran avance a la hora de permitir comunicaciones multilingües a nivel mundial. Sin embargo, pronto aparecieron dos obstáculos importantes. En primer lugar, el número de patrones de bits adicionales disponibles en el código ASCII ampliado es simplemente insuficiente para representar el alfabeto de muchos idiomas asiáticos y de algunos del este de Europa. En segundo lugar, puesto que cada documento estaba restringido a utilizar símbolos en uno solo de los estándares seleccionados, no podían emplearse documentos que contuvieran textos pertenecientes a grupos de idiomas distintos. Ambos obstáculos resultaron ser bastante más graves de lo previsto a la hora de permitir el uso internacional de la tecnología de computadoras. Para resolver estos problemas, se desarrolló el código **Unicode** mediante la cooperación de varios de los principales fabricantes de hardware y software; dicho código ha obtenido rápidamente el respaldo del sector informático. Este código utiliza un patrón distintivo de 16 bits para representar cada símbolo. Como resultado, Unicode

Figura 1.13 El mensaje “Hello.” en ASCII.

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

El Instituto Nacional Estadounidense de Normalización

El instituto ANSI (American National Standards Institute) fue fundado en 1918 por un pequeño consorcio de sociedades de ingeniería y organismos gubernamentales, como una federación sin ánimo de lucro dedicada a coordinar el desarrollo de estándares voluntarios en el sector privado. Hoy día, ANSI tiene como miembros a más de 1300 empresas, organizaciones profesionales, asociaciones empresariales y organismos gubernamentales. La sede de ANSI se encuentra en Nueva York y representa a Estados Unidos como organización miembro de ISO. El sitio web del instituto ANSI es http://www.ansi.org.

En otros países existen organizaciones similares, como por ejemplo Estándares Australia (Standards Australia) en Australia, el Consejo de Estandarización de Canadá (Standards Council of Canada) en Canadá, la Oficina Estatal China para la Calidad y Supervisión Técnica (China State Bureau of Quality and Technical Supervision) en China, el Instituto Alemán de Normalización (Deutsches Institut für Normung) en Alemania, el Comité Japonés de Estándares Industriales (Japanese Industrial Standards Committee) en Japón, la Dirección General de Normas (México), el Comité Estatal de la Federación Rusa para la Estandarización y la Metrología (State Committee of the Russian Federation for Standardization and Metrology) en Rusia, la Asociación Suiza de Normalización (Swiss Association for Standardization) en Suiza, la Institución Británica de Normalización (British Standards Institution) en el Reino Unido, y la Asociación Española de Normalización y Certificación (España).

está compuesto por 65.536 patrones de bits diferentes, lo cual es suficiente para poder escribir textos en idiomas como el chino, el japonés y el hebreo.

Un archivo compuesto por una larga secuencia de símbolos codificados mediante ASCII o Unicode suele denominarse **archivo de texto**. Es importante distinguir entre los archivos de texto simples que son manipulados mediante programas de utilidad denominados **editores de textos** (o simplemente editores), y los archivos más elaborados generados por **procesadores de textos** tales como Microsoft Word. Ambos están compuestos por texto; sin embargo, un archivo de texto contiene únicamente una codificación carácter a carácter del texto, mientras que un archivo generado por un procesador de textos contiene numerosos códigos propietarios que representan cambios en los tipos de fuente, información acerca de la alineación, etc.

Representación de valores numéricos

Almacenar la información en términos de caracteres codificados es poco eficiente cuando la información que se quiere almacenar es puramente numérica. Para ver por qué, considere el problema de almacenar el valor 25. Si insistimos en almacenarlo mediante símbolos codificados en ASCII utilizando un byte por símbolo, necesitaremos un total de 16 bits. Además, el número mayor que podemos almacenar utilizando 16 bits es 99. Sin embargo, como pronto veremos, utilizando **notación binaria** podemos almacenar cualquier entero comprendido dentro del rango que va de 0 a 65535 en esos 16 bits. Por tanto, la notación binaria (o las variaciones de la misma) se emplea ampliamente para codificar datos numéricos de cara a almacenarlos en una computadora.

La notación binaria es una forma de representar valores numéricos utilizando solo los dígitos 0 y 1, en lugar de los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9, como en el sistema tradicional decimal o en base diez. Estudiaremos el sistema binario más en profundidad en la Sección 1.5. Por ahora, lo único que necesitamos es una comprensión elemental de este sistema. Con este objetivo, piense en uno de los anticuados odómetros para vehículos, cuya pantalla contiene únicamente los dígitos 0 y 1, en lugar de los dígitos tradicionales de 0 a 9. El odómetro muestra inicialmente una lectura de todo 0s y a medida que el vehículo recorre los primeros kilómetros, la rueda de más a la derecha gira de 0 a 1. Después, cuando el 1 vuelve de nuevo a 0, hace que aparezca un 1 a su izquierda, generando el patrón 10. El 0 de la derecha gira entonces a 1, generando el patrón 11. Ahora, la rueda situada más a la derecha gira de nuevo de 1 a 0, haciendo que el 1 situado a su izquierda gire a la posición 0 también. Esto hace que aparezca otro 1 en la tercera columna, generando el patrón 100. En resumen, a medida que vamos conduciendo podemos ver la siguiente secuencia de lecturas en el odómetro:

```
0000
0001
0010
0011
0100
0101
0110
0111
1000
```

Esta secuencia está compuesta por las representaciones binarias de los enteros de cero hasta ocho. Aunque sería tedioso, podríamos ampliar esta técnica de recuento para descubrir que el patrón de bits compuesto por dieciséis 1s representa el valor 65535, lo que confirma nuestra afirmación de que podemos codificar con 16 bits cualquier entero perteneciente al rango comprendido entre 0 y 65535.

Debido a esta eficiencia, es habitual almacenar la información numérica con algún tipo de notación binaria en lugar de mediante símbolos codificados. Decimos “con algún tipo de notación binaria”, porque el sistema binario que acabamos de describir es simplemente la base para diversas técnicas de almacena-

ISO— Organización Internacional de Estandarización

La Organización Internacional de Estandarización (más comúnmente denominada ISO) fue establecida en 1947 como una federación mundial de organismos de estandarización, uno por cada país. Actualmente, tiene su sede en Ginebra, Suiza, y tiene más de 100 organizaciones miembro, así como numerosos miembros correspondientes (un miembro correspondiente es normalmente una organización de estandarización de un país que no dispone de un organismo de estandarización reconocido a nivel nacional. Dichos miembros no pueden participar directamente en el desarrollo de estándares pero se les mantiene informados de las actividades de ISO). ISO mantiene un sitio web en la dirección http://www.iso.org.

miento numérico utilizadas en las máquinas actuales. Posteriormente en el capítulo hablaremos de algunas variantes del sistema binario. Por ahora, vamos a limitarnos a decir que comúnmente se emplea un sistema conocido como notación en **complemento a dos** (véase la Sección 1.6) para almacenar los números enteros, porque proporciona un método cómodo de representación de números tanto negativos como positivos. Para representar números con parte fraccionaria, como $4\frac{1}{2}$ o $\frac{3}{4}$ se emplea otra técnica conocida como notación en **punto (coma) flotante** (véase la Sección 1.7).

Representación de imágenes

Un método para representar una imagen consiste en interpretar dicha imagen como una colección de puntos, cada uno de los cuales se denomina **píxel**, abreviatura de la expresión inglesa "*picture element*" (elemento de imagen). A continuación se codifica la apariencia de cada píxel y la imagen completa se representa como una colección de píxeles codificados. A este tipo de colección se le conoce como **mapa de bits**. Esta técnica es muy popular porque muchos dispositivos de visualización, como las impresoras y las pantallas de computadora operan basándose en el concepto de píxel. A su vez, las imágenes en formato de mapa de bits pueden formatearse fácilmente para su visualización.

El método mediante el que se codifican los pixeles que componen un mapa de bits varía de una aplicación a otra. En el caso de una imagen simple en blanco y negro, cada píxel puede representarse mediante un único bit, cuyo valor dependerá de si el píxel correspondiente es negro o blanco. Esta es la técnica empleada en la mayoría de las máquinas de fax. Para fotografías en blanco y negro más elaboradas, cada píxel se puede representar mediante una colección de bits (normalmente ocho), lo que permite representar diversas tonalidades de gris.

En el caso de imágenes en color, cada píxel se codifica mediante un sistema más complejo. Existen dos enfoques bastante comunes. En uno de ellos, que denominaremos codificación RGB, cada píxel se representa mediante tres componentes de color: una componente roja, una componente verde y otra azul, lo que se corresponde con los tres colores primarios de la luz. Para representar la intensidad de cada componente de color normalmente se usa un byte. En consecuencia, hacen falta tres bytes de almacenamiento para representar un único píxel de la imagen original.

Una alternativa a la codificación RGB simple consiste en utilizar una componente de "brillo" y dos componentes de color. En este caso, la componente de "brillo", que se denomina luminancia del píxel, es normalmente la suma de las componentes roja, verde y azul. (De hecho, se considera que es la cantidad de luz blanca del píxel, pero no tenemos por qué preocuparnos de estos detalles aquí.) Las otras dos componentes, denominadas crominancia azul y crominancia roja, se determinan calculando la diferencia entre la luminancia del píxel y la cantidad de luz azul o roja de ese píxel, respectivamente. Juntas, estas tres componentes contienen la información necesaria para reproducir el píxel.

La popularidad de las imágenes codificadas utilizando las componentes de luminancia y crominancia proviene del campo de las emisiones en televisión en color, porque esta técnica proporciona un método de codificar las imágenes en color que era también compatible con los antiguos receptores de televisión

en blanco y negro. De hecho, podemos obtener una versión en escala de grises de una imagen utilizando únicamente las componentes de luminancia de la imagen en color codificada.

Una desventaja de representar las imágenes en mapas de bits es que no puede cambiarse fácilmente la escala de la imagen para que esta adopte cualquier tamaño que deseemos. Básicamente, la única forma de agrandar la imagen es haciendo los píxeles más grandes, lo que hace que la imagen termine por tener una apariencia granulada. (Esta es la técnica conocida como "zoom digital", empleada en las cámaras digitales, en oposición al "zoom óptico", que se obtiene ajustando el objetivo de la cámara.)

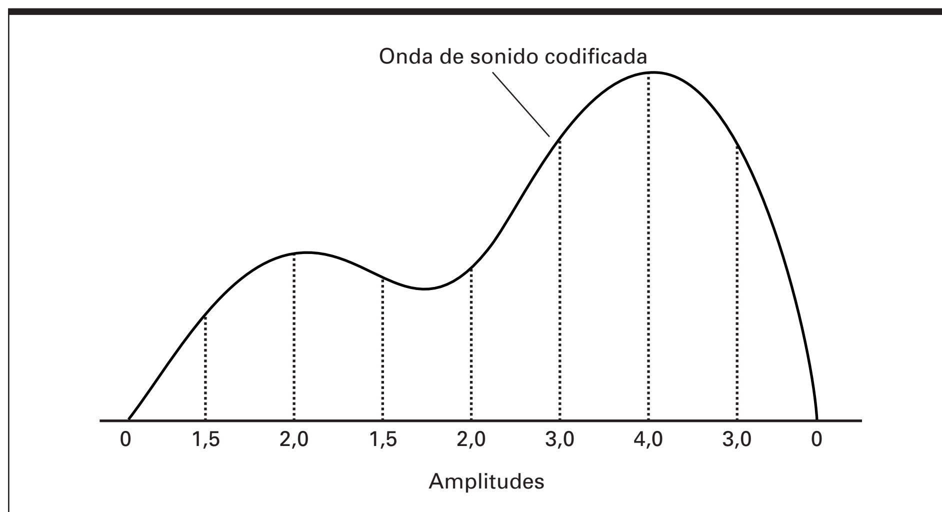
Una forma alternativa de representar imágenes que evita estos problemas de cambio de escala consiste en describir la imagen como una colección de estructuras geométricas, tales como líneas y curvas, que pueden codificarse mediante técnicas de geometría analítica. Esta descripción permite al dispositivo que vaya a mostrar la imagen decidir cómo deben visualizarse las estructuras geométricas, en lugar de insistir en que ese dispositivo reproduzca un patrón de píxeles concreto. Esta es la técnica utilizada para generar los tipos de fuente escalables que hay disponibles hoy día en los sistemas de procesamiento de textos. Por ejemplo, TrueType (desarrollado por Microsoft y Apple) es un sistema para describir geométricamente símbolos de texto. De forma similar, PostScript (desarrollado por Adobe Systems) proporciona un medio de describir caracteres, así como datos de imágenes más generales. Esta forma geométrica de representar las imágenes es también muy popular en los sistemas de **diseño asistido por computadora (CAD, *Computer-aided design*)**, en los que se visualizan y manipulan en las pantallas de computadora dibujos de objetos tridimensionales.

La diferencia entre representar una imagen mediante estructuras geométricas y representarla mediante mapas de bits es evidente para los usuarios de muchos sistemas software para dibujo de imágenes (como la aplicación Paint de Microsoft) que permiten al usuario dibujar imágenes compuestas por formas preestablecidas tales como rectángulos, óvalos y curvas elementales. El usuario simplemente selecciona la forma geométrica deseada en un menú y luego controla el dibujo de esa forma mediante el ratón. Durante el proceso de dibujo, el software mantiene una descripción geométrica de la forma que se está dibujando. A medida que se proporcionan instrucciones a través del ratón, la representación geométrica interna se modifica, se reconvierte a formato de mapa de bits y se visualiza. Esto permite cambiar fácilmente la escala y la forma de la imagen. Sin embargo, una vez que se ha completado el proceso de dibujo, la descripción geométrica subyacente se descarta y solo queda el mapa de bits, lo que quiere decir que todas las modificaciones posteriores requieren un proceso de ajuste muy tedioso píxel a píxel. Por el contrario, otros sistemas de dibujo conservan la descripción de las formas geométricas permitiendo su modificación posteriormente. Con estos sistemas, puede cambiarse fácilmente el tamaño de esas formas, manteniendo una visualización perfecta independientemente del tamaño de cada forma.

Representación de sonidos

El método más genérico de codificar información de audio para su almacenamiento y manipulación en una computadora consiste en muestrear la amplitud

Figura 1.14 Onda de sonido representada por la secuencia 0, 1,5; 2,0; 1,5; 2,0; 3,0; 4,0; 3,0; 0.



de la onda de sonido a intervalos regulares y grabar la serie de valores obtenidos. Por ejemplo, la serie 0, 1,5, 2,0, 1,5, 2,0, 3,0, 4,0, 3,0, 0 representaría una onda sonora cuya amplitud aumenta, disminuye brevemente, aumenta a un nivel superior y luego vuelve a caer a 0 (Figura 1.14). Esta técnica, empleando una tasa de muestreo de 8000 muestras por segundo, se ha estado utilizando durante años en las comunicaciones telefónicas de voz a larga distancia. La voz en un extremo del canal de comunicación se codifica como una serie de valores numéricos que representan la amplitud de la voz en cada intervalo de 125 microsegundos. Estos valores numéricos se transmiten luego a través de la línea de comunicación hasta el extremo receptor, donde se utilizan para reproducir el sonido de la voz.

Aunque 8000 muestras por segundo puede parecer una tasa muy rápida, no resulta suficiente para la grabación de música en alta fidelidad. Para obtener una reproducción de sonido de calidad comparable a la que puede obtenerse con los CD musicales de hoy día, se utiliza una tasa de muestreo de 44.100 muestras por segundo. Los datos obtenidos para cada muestra se presentan mediante 16 bits (32 bits para grabaciones estéreo). En consecuencia, cada segundo de música almacenado en estéreo requiere más de un millón de bits.

Un sistema de codificación alternativo, conocido con el nombre de MIDI (*Musical Instrument Digital Interface*, Interfaz digital para instrumentos musicales), se utiliza de forma bastante común en los sintetizadores musicales usados en los órganos electrónicos, así como también en la generación de sonidos para videojuegos y en los efectos sonoros empleados en algunos sitios web. Codificando las instrucciones para la generación de música en un sintetizador en lugar de codificar el propio sonido, MIDI no tiene unos requisitos de almacenamiento tan grandes como la técnica de muestreo. Para ser más precisos, MIDI codifica qué instrumento debe reproducir cada nota y durante cuánto tiempo, lo que quiere decir que un clarinete que esté tocando la nota Sol durante dos segundos puede codificarse con tres bytes, en lugar de requerir los más de dos millones de bits si muestreamos el sonido a una tasa de 44.100 muestras por segundo.

Resumiendo, MIDI puede considerarse como una forma de codificar las partituras que lee un intérprete, en lugar de codificar la propia interpretación. A su vez, eso implica que una "grabación" MIDI puede sonar de forma significativamente distinta cuando se reproduce en diferentes sintetizadores.

Cuestiones y ejercicios

1. He aquí un mensaje codificado en ASCII utilizando 8 bits por símbolo. ¿Qué dice el mensaje? (*Véase* el Apéndice A).

```
01000011 01101111 01101101 01110000 01110101 01110100
01100101 01110010 00100000 01010011 01100011 01101001
01100101 01101110 01100011 01100101
```

2. En el código ASCII, ¿cuál es la relación entre los códigos correspondientes a una letra mayúscula y a la misma letra pero minúscula? (*Véase* el Apéndice A.)
3. Codifique en ASCII las siguientes frases:

   a. "Stop!" Cheryl shouted. **b.** Does 2 + 3 = 5?

4. Describa un dispositivo de la vida cotidiana que pueda encontrarse en uno de dos estados posibles, como por ejemplo una bombilla que puede estar encendida o apagada. Asigne el símbolo 1 a uno de esos estados y 0 al otro y muestre cómo aparecería la representación ASCII de la letra *b* cuando se la almacenara con tales bits.
5. Convierta cada una de las siguientes representaciones binarias a su formato equivalente en base diez:

   a. 0101 **b.** 1001 **c.** 1011
   d. 0110 **e.** 10000 **f.** 10010

6. Convierta cada una de las siguientes representaciones en base diez a su formato equivalente en binario:

   a. 6 **b.** 13 **c.** 11
   d. 18 **e.** 27 **f.** 4

7. ¿Cuál es el valor mayor numérico que podría representarse con tres bytes si codificáramos cada dígito utilizando un patrón ASCII de un byte? ¿Y cuál sería en caso de utilizar notación binaria?
8. Una alternativa a la notación hexadecimal para representar patrones de bits es la **notación decimal con punto**, en la que cada byte del patrón se representa mediante su valor equivalente en base diez. A su vez, estas representaciones de bytes se separan mediante puntos. Por ejemplo, 12.5 representa el patrón 0000110000000101 (el byte 00001100 se representa mediante 12 y 00000101 se representa mediante 5), y el patrón 100010000001000000000111 se representa mediante 136.16.7. Represente cada uno de los siguientes patrones de bits en notación decimal con puntos.

a. 0000111100001111 **b.** 0011001100000000010000000
c. 0000101010100000

9. Cite una ventaja de representar las imágenes mediante estructuras geométricas, en lugar de mediante mapas de bits. ¿Qué ventaja tienen las técnicas de mapa de bits frente a las basadas en estructuras geométricas?

10. Suponga que codificamos una grabación estéreo de una hora de música utilizando una tasa de muestreo de 44.100 muestras por segundo como se explica en el texto. Compare el tamaño de la versión codificada con la capacidad de almacenamiento de un CD.

1.5 El sistema binario

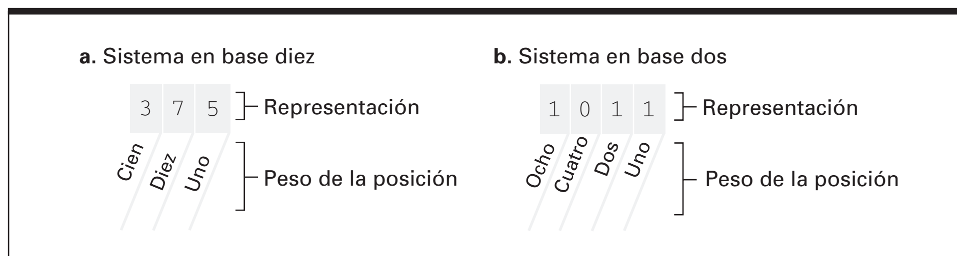
En la Sección 1.4 hemos visto que la notación binaria es una forma de representar valores numéricos utilizando solo los dígitos 0 y 1, en lugar de los diez dígitos 0 a 9 que se emplean en el sistema más común en notación en base diez. Vamos a echar ahora un vistazo más detallado a la notación binaria.

Notación binaria

Recuerde que en el sistema en base diez, cada posición de una representación numérica está asociada con un determinado peso. En la representación 375, el 5 se encuentra en la posición asociada con el peso uno, el 7 está en la posición asociada con el peso diez y el 3 está en la posición asociada con el peso cien (Figura 1.15a). Cada uno de esos pesos es diez veces el peso de la posición situada a su derecha. El valor representado por la expresión completa se obtiene multiplicando el valor de cada dígito por el peso asociado con la posición de este dígito y luego sumando esos productos. Para ilustrar el proceso, el patrón 375 representa (3 × cien) + (7 × diez) + (5 × uno), lo que en notación más técnica sería $(3 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$.

La posición de cada dígito en notación binaria también está asociada con un peso, pero el peso de cada posición es igual al doble del peso asociado con la posición de su derecha. Para ser más precisos, el dígito más a la derecha en una representación binaria está asociado con uno (2^0), la siguiente posición a la izquierda está asociada con dos (2^1), la siguiente está asociada con

Figura 1.15 Sistema en base diez y sistema binario.



cuatro (2^2), la siguiente con ocho (2^3), y así sucesivamente. Por ejemplo, en la representación binaria 1011, el 1 de más a la derecha está en la posición de peso uno, el 1 situado a continuación está en la posición de peso dos, el 0 se encuentra en la posición de peso cuatro y el 1 más a la izquierda está en la posición de peso ocho (Figura 1.15b).

Para obtener el valor correspondiente a una representación binaria, seguimos el mismo procedimiento que en base diez: multiplicamos el valor de cada dígito por el peso asociado con su posición y sumamos los resultados. Por ejemplo, el valor representado por 100101 es 37, como se muestra en la Figura 1.16. Observe que, como la notación binaria solo utiliza los dígitos 0 y 1, este proceso de multiplicación y suma se reduce simplemente a sumar los pesos de las posiciones que están ocupadas por 1s. Por tanto, el patrón binario 1011 representa el valor once, porque los 1s se encuentran en las posiciones de pesos uno, dos y ocho.

En la Sección 1.4 hemos aprendido a contar en notación binaria, lo que nos ha permitido codificar enteros de pequeño tamaño. Para calcular las representaciones binarias de valores grandes es preferible emplear la técnica descrita por el algoritmo de la Figura 1.17. Apliquemos este algoritmo al valor trece (Figura 1.18). Dividimos primero trece entre dos, obteniendo como cociente seis y como resto uno. Dado que el cociente no es cero, el Paso 2 nos dice que hay que dividir el cociente (seis) entre dos, obteniendo como nuevo cociente tres y un resto igual a cero. El nuevo cociente sigue sin ser cero, por lo que lo dividimos entre dos, obteniendo como cociente uno y como resto uno. Una vez más, dividimos el último cociente (uno) entre dos, obteniendo esta vez un cociente de cero y un

Figura 1.16 Decodificación de la representación binaria 100101.

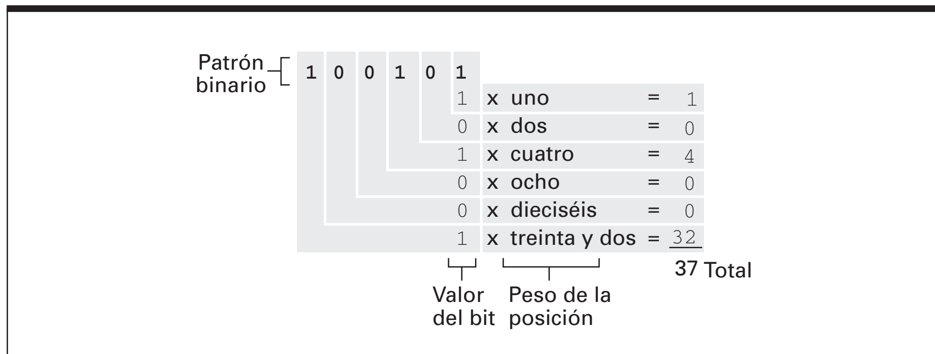


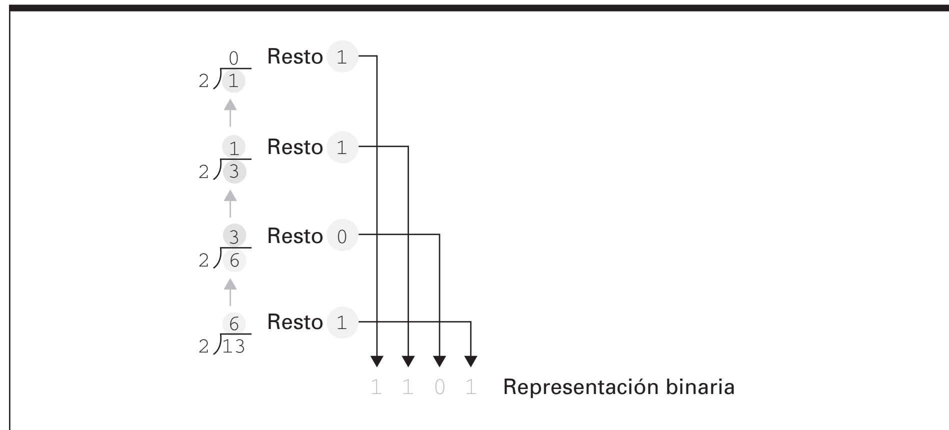
Figura 1.17 Algoritmo para determinar la representación binaria de un entero positivo.

Paso 1. Dividir el valor entre dos y anotar el resto.

Paso 2. Mientras que el cociente obtenido sea distinto de cero, continuar dividiendo entre dos el último cociente y anotar el resto.

Paso 3. Ahora que ha obtenido un cociente igual a cero, la representación binaria del valor original está compuesta por los restos obtenidos, ordenados de derecha a izquierda según el orden en que fueron anotados.

Figura 1.18 Aplicación del algoritmo de la Figura 1.17 para otener la representación binaria de trece.



resto igual a uno. Puesto que ya tenemos un cociente igual a cero, continuamos con el Paso 3, donde vemos que la representación binaria del valor original (trece) es 1101, obtenida a partir de la lista de restos.

Suma binaria

Para comprender el proceso de sumar dos enteros representados en binario, recordemos primero el proceso de suma de valores representados en notación tradicional en base diez. Considere, por ejemplo, el siguiente problema:

```
  58
+ 27
----
```

Comenzamos sumando el 8 y el 7 de la columna situada más a la derecha para obtener la suma 15. Escribimos un 5 debajo de dicha columna y nos llevamos el 1 a la columna siguiente, obteniendo

```
  1
  58
+ 27
----
   5
```

Ahora sumamos el 5 y el 2 de la columna siguiente junto con el 1 que hemos acarreado obteniéndose como suma 8, que será el valor que escribamos debajo de la columna. El resultado es el siguiente:

```
  58
+ 27
----
  85
```

En resumen, el procedimiento consiste en ir avanzando de derecha a izquierda mientras vamos sumando los dígitos de cada columna escribiendo el dígito menos significativo de dicha suma bajo la columna en cuestión y acarreando el dígito más significativo de la suma (si es que hay uno) a la columna siguiente.

Para sumar dos enteros representados en notación binaria, seguimos el mismo procedimiento salvo porque las sumas se realizan ahora usando las reglas de la suma que se muestran en la Figura 1.19, en lugar de las reglas tradicionales en base diez que hemos aprendido en la escuela elemental. Por ejemplo, para resolver el problema

```
  111010
+  11011
```

comenzamos sumando los valores 0 y 1 de más a la derecha; obtenemos 1, que escribimos debajo de la columna. A continuación, sumamos el 1 y el 1 de la columna siguiente, obteniendo 10. Escribimos el 0 de ese 10 debajo de la columna y acarreamos el 1 a la parte superior de la columna siguiente. En este punto, nuestra solución tiene el aspecto siguiente:

```
     1
  111010
+  11011
      01
```

Sumamos el 1, el 0 y el 0 de la siguiente columna y obtenemos 1, por lo que escribimos 1 bajo dicha columna. El 1 y el 1 de la columna siguiente nos dan un resultado igual a 10; escribimos el 0 bajo la columna y acarreamos el 1 a la siguiente columna. Ahora la solución tiene este aspecto:

```
   1
  111010
+  11011
    0101
```

El 1, el 1 y el 1 de la siguiente columna suman 11 (equivalente binario del valor tres); escribimos el 1 de menor peso bajo la columna y acarreamos el otro 1 a la parte superior de la siguiente columna. Sumamos dicho 1 al 1 ya existente en dicha columna para obtener 10. De nuevo, escribimos el 0 de menor peso y acarreamos el 1 a la columna siguiente. Ahora tenemos

```
 1
  111010
+  11011
  010101
```

El único valor presente en la columna siguiente es el 1 que hemos acarreado de la columna anterior, así que los escribimos en la respuesta. La solución final será:

```
  111010
+  11011
 1010101
```

Figura 1.19 Reglas de la suma binaria.

```
 0    1    0    1
+0   +0   +1   +1
 0    1    1   10
```

Tecnologías analógica y digital

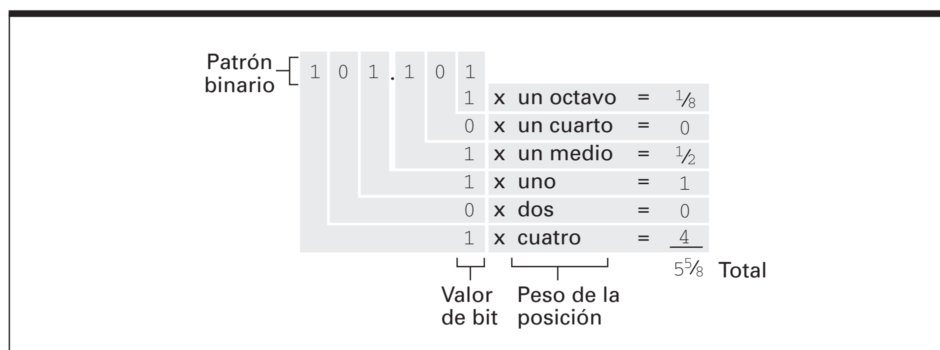
Antes de la llegada del siglo XXI, muchos investigadores dedicaron su esfuerzo a debatir los pros y los contras de la tecnología digital por comparación con la analógica. En un sistema digital, cada valor se codifica mediante una serie de dígitos y luego se almacena utilizando varios dispositivos, cada uno de los cuales representará uno de los dígitos. En un sistema analógico, cada valor se almacena en un único dispositivo que puede representar un valor comprendido dentro de un rango continuo.

Compararemos las dos técnicas utilizando cubos de agua como medios de almacenamiento. Para simular un sistema digital, podríamos acordar que un cubo vacío represente el dígito 0 y que un cubo lleno represente el dígito 1. Entonces, podríamos almacenar un valor numérico en una fila de cubos utilizando notación en punto flotante (véase la Sección 1.7). Por contraste, podríamos simular un sistema analógico llenando parcialmente un único cubo hasta el punto en el que el nivel de agua se corresponda con el valor numérico que deseamos representar. A primera vista, puede parecer que el sistema analógico es más preciso, ya que no sufrirá los errores de truncamiento inherentes a los sistemas digitales (consulte de nuevo la Sección 1.7). Sin embargo, cualquier movimiento del cubo en el sistema analógico podría provocar errores a la hora de detectar el nivel del agua, mientras que haría falta que se vertiera mucha agua en el sistema digital antes de que empezara a difuminarse la distinción entre un cubo vacío y un cubo lleno. Por tanto, el sistema digital será menos sensible a los errores que el sistema analógico. Esta robustez es una de las principales razones por la que muchas aplicaciones que originalmente estaban basadas en tecnología analógica (como por ejemplo las comunicaciones telefónicas, las grabaciones audio y las emisiones de televisión) están efectuando la migración hacia la tecnología digital.

Números fraccionarios en binario

Para ampliar la notación binaria con el fin de admitir valores fraccionarios, utilizamos un **punto de separación** que cumple el mismo papel que la coma decimal en la notación decimal. Es decir, los dígitos situados a la izquierda de la coma representan la parte entera del valor y se interpretan como en el sistema binario del que hemos hablado anteriormente. Los dígitos situados a la derecha representan la parte fraccionaria del valor y se interpretan de forma similar a los otros bits, salvo porque a sus posiciones se les asignan valores fraccionarios. Es decir, la primera posición situada a la derecha del punto de separación tendrá asignado el peso ½ (que es 2^{-1}), la siguiente posición tendrá de peso ¼ (que es 2^{-2}), la siguiente tendrá de peso ⅛ (que es 2^{-3}), y así sucesivamente. Observe que se trata simplemente de una continuación de la regla que hemos enunciando antes: a cada posición se le asigna un peso que es igual a dos veces el peso del bit de su derecha. Habiéndose asignado estos pesos a las posiciones de cada bit, la decodificación de una representación binaria que contenga un punto de separación requiere el mismo procedimiento que ya hemos utilizado sin dicho punto. Para ser más precisos, multiplicamos cada valor de bit por el peso de la posición de dicho bit en la representación. Por ejemplo, la representación binaria 101.101 se decodificará como 5 ⅝, como se muestra en la Figura 1.20.

Figura 1.20 Decodificación de la representación binaria 101.101



Para la suma, las técnicas que se aplican con el sistema en base diez también son aplicables en binario. Por ejemplo, para sumar dos representaciones binarias con su correspondiente punto de separación, basta con alinear los puntos de separación y aplicar el mismo proceso de suma que antes. Por ejemplo, si sumamos 10.011 con 100.11 obtenemos 111.001, como aquí se muestra:

```
   10.011
+ 100.110
---------
  111.001
```

Cuestiones y ejercicios

1. Convierta cada una de las siguientes representaciones binarias a su forma equivalente en base diez:

a. 101010 **b.** 100001 **c.** 10111 **d.** 0110 **e.** 11111

2. Convierta cada una de las siguientes representaciones en base diez a su forma binaria equivalente:

a. 32 **b.** 64 **c.** 96 **d.** 15 **e.** 27

3. Convierta cada una de las siguientes representaciones binarias a su forma equivalente en base diez:

a. 11.01 **b.** 101.111 **c.** 10.1 **d.** 110.011 **e.** 0.101

4. Exprese los siguientes valores en notación binaria:

a. $4\frac{1}{2}$ **b.** $2\frac{3}{4}$ **c.** $1\frac{1}{8}$ **d.** $\frac{5}{16}$ **e.** $5\frac{5}{8}$

5. Realice las siguientes sumas en notación binaria:

a.
```
 11011
+1100
------
```
b.
```
  1010.001
+    1.101
----------
```
c.
```
  11111
+ 0001
-------
```
d.
```
  111.11
+ 00.01
--------
```

1.6 Almacenamiento de enteros

Los matemáticos han estado interesados desde hace mucho tiempo en los sistemas de notación numérica y muchas de sus ideas han resultado ser bastante compatibles con el diseño de circuitos digitales. En esta sección vamos a considerar dos de estos sistemas de notación, la notación en complemento a dos y la notación en exceso, que se emplean para representar valores enteros en los equipos de computación. Estos sistemas están basados en el sistema binario, pero tienen propiedades adicionales que los hacen más compatibles con el diseño de computadoras. Sin embargo, estas ventajas también tienen asociadas ciertas desventajas. Nuestro objetivo es comprender estas propiedades y ver cómo afectan a la utilización de las computadoras.

Notación en complemento a dos

El sistema más popular para la representación de enteros en las computadoras actuales es la notación en **complemento a dos**. Este sistema utiliza un número fijo de bits para representar cada uno de los valores del sistema. En los equipos actuales, es habitual utilizar un sistema de complemento a dos en el que cada valor está representado por un patrón de 32 bits. Un sistema de tamaño tan grande permite representar un amplio rango de números, pero resulta bastante incómodo para la realización de ejemplos. Por tanto, de cara a estudiar las propiedades de los sistemas en complemento a dos, nos vamos a centrar en sistemas de menor tamaño.

En la Figura 1.21 se muestran dos sistemas completos de complemento a dos: uno basado en patrones de bits de longitud igual a tres y el otro basado en patrones de bits de longitud igual a cuatro. Un sistema como estos se construye partiendo de una cadena de 0s de la longitud apropiada y luego contando en binario hasta alcanzar el patrón formado por un único 0 seguido de una serie de 1s. Estos patrones representan los valores 0, 1, 2, 3,... Los patrones que representan los valores negativos se obtienen comenzando por una cadena de 1s de la longitud apropiada y luego contando hacia abajo en binario hasta obtener el patrón compuesto por un único 1 seguido de 0s. Estos patrones representan los valores −1, −2, −3,... (Si el contar hacia atrás en binario le resulta difícil, comience por la parte inferior de la tabla con el patrón compuesto por un único 1 seguido de 0s, y cuente hacia arriba hasta alcanzar el patrón formado por todo 1s.)

Observe que en un sistema de complemento a dos, el bit de más a la izquierda de cada patrón de bits indica el signo del valor representado. Por ello, el bit de más a la izquierda se suele denominar **bit de signo**. En un sistema en complemento a dos, los valores negativos se representan mediante patrones cuyo bit de signo es igual a 1; los valores no negativos se representan mediante patrones cuyo bit de signo es igual a 0.

En un sistema en complemento a dos, existe una relación bastante conveniente entre los patrones que representan los valores positivos y negativos que tienen el mismo módulo. Dicha relación es la siguiente: ambos patrones son idénticos cuando se los lee de derecha a izquierda, hasta llegar al primer 1. A partir de ahí, los patrones son complementarios el uno del otro. (El **complemento** de un patrón es el patrón que se obtiene cambiando todos los 0s por 1s y todos los 1s por 0s, 0110 y 1001 son complementarios entre sí.) Por

Figura 1.21 Sistemas de notación en complemento a dos.

a. Con patrones de longitud igual a tres

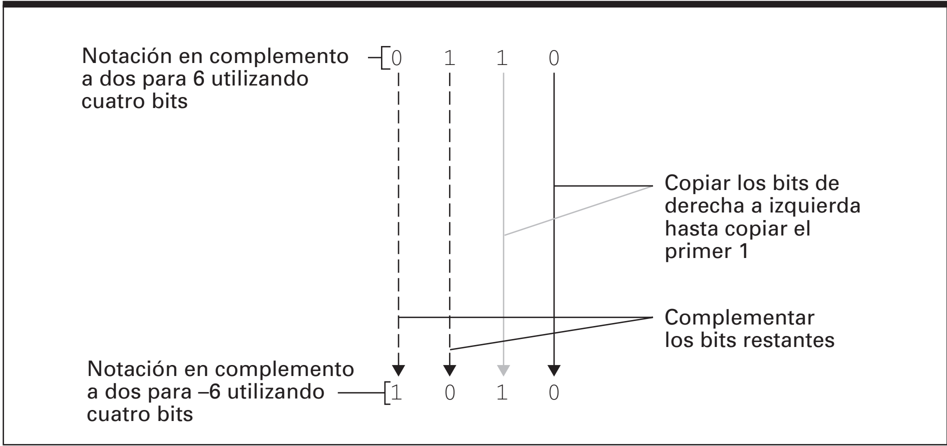
Patrón de bits	Valor representado
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Con patrones de longitud igual a cuatro

Patrón de bits	Valor representado
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

ejemplo, en el sistema de 4 bits de la Figura 1.21 los patrones que representan los valores 2 y −2 terminan ambos en 10, pero el patrón que representa el valor 2 comienza por 00, mientras que el patrón representa el valor −2 comienza por 11. Esta observación nos lleva a un algoritmo para realizar la conversión entre los patrones de bits que representan los valores positivo y negativo de un mismo módulo. Basta con copiar el patrón original de derecha a izquierda hasta copiar el primer 1 y luego complementar los bits restantes antes de transferirlos al patrón final de bits (Figura 1.22).

Figura 1.22 Codificación del valor −6 en notación en complemento a dos usando 4 bits.



Comprender estas propiedades básicas de los sistemas en complemento a dos nos permite obtener también un algoritmo para decodificar las representaciones en complemento a dos. Si el patrón que hay que decodificar tiene un bit de signo igual a 0, lo único que tendremos que hacer es leer el valor como si el patrón fuera una representación binaria normal y corriente. Por ejemplo, 0110 representa el valor 6, porque 110 es el equivalente binario del valor decimal 6. Si el patrón que hay que decodificar tiene un bit de signo igual a 1, sabemos que el valor representado es negativo y lo único que necesitaremos es encontrar el módulo de dicho valor. Para ello, aplicaremos el procedimiento de "copiar y complementar" mostrado en la Figura 1.22 y luego decodificaremos el patrón obtenido como si fuera una representación binaria normal y corriente. Por ejemplo, para decodificar el patrón 1010, primero vemos que el valor representado es negativo porque el bit de signo es 1. Por ello, aplicamos el procedimiento de "copia y complemento" para obtener el patrón 0110, pudiendo comprobar que se trata de la representación binaria del valor 6. De este modo, concluimos que el patrón original representa el valor −6.

Suma utilizando la notación en complemento a dos Para sumar valores representados en notación de complemento a dos aplicamos el mismo algoritmo que ya hemos empleado anteriormente para la suma binaria, salvo por el hecho de que todos los patrones de bits, incluyendo la respuesta, tendrán la misma longitud. Esto quiere decir que cuando se realiza la suma en un sistema de complemento a dos, será preciso truncar cualquier bit adicional generado a la izquierda de la respuesta por alguna operación final de acarreo. Por tanto, la "suma" de 0101 y de 0010 es 0111, mientras que la "suma" de 0111 y 1011 da como resultado 0010 (0111 + 1011 = 10010, que se trunca a 0010).

Teniendo esto presente, considere los tres problemas de suma mostrados en la Figura 1.23. En cada caso, hemos traducido el problema a la notación de complemento a dos (utilizando patrones de bits de longitud igual cuatro), hemos realizado el proceso de suma anteriormente descrito y hemos decodificado el resultado, para obtener el equivalente en nuestra notación usual en base diez.

Observe que el tercer problema de la Figura 1.23 implica la suma de un número positivo y otro negativo, lo que ilustra una de las ventajas principales

Figura 1.23 Problemas de suma convertidos en notación en complemento a dos.

Problema en base diez		Problema en complemento a dos		Respuesta en base diez
3 + 2	→	0011 + 0010 = 0101	→	5
−3 + −2	→	1101 + 1110 = 1011	→	−5
7 + −5	→	0111 + 1011 = 0010	→	2

de la notación en complemento a dos. Podemos realizar la suma de cualquier combinación de números con signos iguales o distintos utilizando el mismo algoritmo y por tanto la misma circuitería. Esto contrasta de manera notable con la forma en que los seres humanos llevamos a cabo normalmente operaciones aritméticas. Mientras que a los niños se les enseña primero en la escuela elemental cómo sumar y solo después se les enseña a restar, una máquina que utilice notación en complemento a dos solo necesita saber cómo sumar.

Por ejemplo, el problema de realizar la resta 7 − 5 es el mismo que el problema de realizar la suma 7 + (−5). En consecuencia, si una máquina tuviera que dar el resultado de restar 5 (almacenado como 0101) de 7 (almacenado como 0111), primero cambiará el 5 por −5 (representado como 1011) y luego llevaría a cabo el proceso para sumar 0111 + 1011 obteniendo 0010, que representa el valor decimal 2, de la manera siguiente:

```
  7            0111           0111
 −5    →     − 0101    →    + 1011
                              0010    →    2
```

Vemos que cuando se utiliza la notación de complemento a dos para representar valores numéricos, basta con un circuito de suma combinado con un circuito que determine el valor negativo de un valor dado para así resolver problemas de suma y resta. (Tales circuitos se muestran y explican en el Apéndice B.)

El problema del desbordamiento Un problema que hemos evitado en los ejemplos anteriores es que en todo sistema de complemento a dos existe un límite para el tamaño de los valores que se pueden representar. Si utilizamos complemento a dos con patrones de 4 bits, el entero positivo mayor que es posible representar es 7 y el entero negativo más grande es −8. En particular, no podemos representar el valor 9, lo que significa que no podemos esperar obtener la respuesta correcta al problema de sumar 5 + 4. De hecho, el resultado que obtendríamos sería −7. Este fenómeno se conoce con el nombre de **desbordamiento**. Es decir, el desbordamiento se produce cuando un cálculo genera un valor que cae fuera del rango de valores que puede ser representado. Cuando se usa la notación de complemento a dos, esto puede ocurrir tanto al sumar dos valores positivos como al sumar dos valores negativos. En cualquiera de los casos, la condición puede detectarse comprobando el bit de signo de la respuesta. Se sabe que existe desbordamiento si la suma de dos valores positivos da como resultado un patrón que corresponde a un valor negativo o si la suma de dos valores negativos da como resultado un valor positivo.

Evidentemente, dado que la mayoría de las computadoras emplean sistemas de complemento a dos con patrones de bits más largos que los que hemos usado en nuestros ejemplos, es posible manipular valores mucho más mayores sin que se produzca desbordamiento. Actualmente, es habitual emplear patrones de 32 bits para almacenar valores en notación de complemento a dos, que permiten acumular valores positivos tan grandes como 2.147.483.647 antes de que se produzca un desbordamiento. Si se necesitaran valores aun más grandes, podrían emplearse patrones de bits más largos o quizá podrían modificarse las unidades de medida. Por ejemplo, determinar una solución en términos de millas en lugar de en términos de pulgadas genera números más pequeños e incluso proporcionaría la precisión requerida.

La cuestión es que las computadoras pueden cometer errores. Por tanto, la persona que utiliza la máquina tiene que ser consciente de los riesgos que esto implica. Un problema de los programadores y usuarios de computadoras es que se vuelven complacientes e ignoran el hecho de que valores pequeños pueden acumularse para generar números grandes. Por ejemplo, hace tiempo era normal utilizar patrones de 16 bits para representar valores en notación de complemento a dos, lo que significaba que el desbordamiento se producía cuando se alcanzaban valores iguales o mayores que $2^{15} = 32.768$. El 19 de septiembre de 1989, un sistema informático hospitalario falló después de haber estado años proporcionando un servicio fiable. Una inspección en profundidad reveló que ese fecha era exactamente 32.768 días después del 1 de enero de 1900, y la máquina estaba programada para calcular las fechas basándose en esa fecha de inicio. Así, debido al desbordamiento, el 19 de septiembre de 1989 generó un valor negativo, un fenómeno para el que la programación de la computadora no estaba diseñada y no sabía cómo manejar.

Notación en exceso

Otro método para representar valores enteros es la **notación en exceso**. Como sucede con la notación en complemento a dos, cada uno de los valores de un sistema de notación en exceso se representa mediante un patrón de bits de la misma longitud. Para establecer un sistema en exceso, primero seleccionamos la longitud del patrón que deseamos utilizar y luego escribimos todos los diferentes patrones de bits de dicha longitud en el orden que aparecerían si estuviéramos contando en binario. A continuación, observamos que el primer patrón con un 1 como el bit más significativo aparece aproximadamente en mitad de la lista. Seleccionaremos este patrón para representar el cero; los siguientes patrones se emplean para representar 1, 2, 3,...; y los patrones que le anteceden se utilizan para −1, −2, −3,... El código resultante, cuando se emplean patrones de longitud igual a cuatro, se muestra en la Figura 1.24. En

Figura 1.24 Tabla de conversión para un sistema exceso ocho.

Patrón de bits	Valor representado
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

Figura 1.25 Sistema de notación en exceso utilizando patrones de bits de longitud tres.

Patrón de bits	Valor representado
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

ella podemos ver que el valor 5 está representado por el patrón 1101 y el valor −5 está representado por 0011. (Observe que la diferencia entre un sistema en exceso y un sistema en complemento a dos es que los bits de signo están invertidos.)

El sistema representado en la Figura 1.24 se conoce con el nombre de notación exceso ocho. Para entender por qué, interpretamos primero cada uno de los patrones que forman el código utilizando el sistema binario tradicional y luego comparamos dichos resultados con los valores representados en la notación en exceso. En cado caso, veremos que la interpretación binaria es ocho unidades superior a la interpretación con notación en exceso, Por ejemplo, el patrón 1100 en notación binaria representa el valor 12, pero en nuestro sistema en exceso representa el valor 4; 0000 en notación binaria representa 0, pero en el sistema en exceso representa el valor −8. De forma similar, un sistema en exceso basado en patrones de longitud cinco se denominaría notación exceso 16, porque el patrón 10000, por ejemplo, se emplearía para representar el cero en lugar de representar el valor usual de 16. De la misma forma, puede comprobar que el sistema en exceso de tres bits se podría denominar notación exceso cuatro (Figura 1.25).

Cuestiones y ejercicios

1. Convierta cada una de las siguientes representaciones en complemento a dos a su forma equivalente en base diez:

a. 00011 **b.** 01111 **c.** 11100
d. 11010 **e.** 00000 **f.** 10000

2. Convierta cada una de las siguientes representaciones en base diez a su forma equivalente en complemento a dos utilizando patrones de 8 bits:

a. 6 **b.** −6 **c.** −17
d. 13 **e.** −1 **f.** 0

3. Suponga que los siguientes patrones de bits representan valores almacenados en complemento a dos. Determine la representación en complemento a dos del negado de cada uno de esos valores:

a. 00000001 **b.** 01010101 **c.** 11111100
d. 11111110 **e.** 00000000 **f.** 01111111

4. Suponga que una máquina almacena números en notación en complemento a dos. ¿Cuáles son los números más grande y más pequeño que pueden almacenarse si la máquina utiliza patrones de bits de las siguientes longitudes?

a. cuatro **b.** seis **c.** ocho

5. En los siguientes problemas, cada patrón de bits representa un valor almacenado con notación en complemento a dos. Calcule la respuesta a cada problema con notación en complemento a dos realizando el proceso de suma descrito en el texto. Después compruebe los resultados traduciendo el problema y la respuesta a notación en base diez.

a.	b.	c.	d.	e.
0101	0011	0101	1110	1010
+ 0010	+ 0001	+ 1010	+ 0011	+ 1110

6. Resuelva cada uno de los problemas siguientes en notación en complemento a dos, pero en esta ocasión fíjese si se producen desbordamientos e indique qué respuestas son incorrectas debido a dicho fenómeno.

a.	b.	c.	d.	e.
0100	0101	1010	1010	0111
+ 0011	+ 0110	+ 1010	+ 0111	+ 0001

7. Traduzca cada uno de los siguientes problemas de notación en base diez a notación en complemento a dos utilizando patrones de bits de longitud cuatro, a continuación convierta cada problema a un problema de suma equivalente (como lo haría una máquina) y realice la suma. Compruebe sus respuestas convirtiéndolas de nuevo a base diez.

a.	b.	c.	d.	e.
6	3	4	2	1
−(−1)	−2	−6	−(−4)	−5

8. ¿Puede producirse desbordamiento al sumar valores en notación de complemento a dos, siendo uno de los valores positivo y el otro negativo? Explique su respuesta.

9. Convierta cada una de las siguientes representaciones exceso ocho a su forma equivalente en base diez, sin consultar la tabla incluida en el texto:

a. 1110 **b.** 0111 **c.** 1000 **d.** 0010 **e.** 0000 **f.** 1001

10. Convierta cada una de las siguientes representaciones en base diez a su forma equivalente exceso ocho, sin consultar la tabla incluida en el texto:

a. 5 **b.** −5 **c.** 3 **d.** 0 **e.** 7 **f.** −8

11. ¿Puede representarse el valor 9 en notación exceso ocho? ¿Y se puede puede representar el valor 6 en notación exceso cuatro? Explique su respuesta.

1.7 Almacenamiento de números fraccionarios

Por contraste con el almacenamiento de enteros, el almacenamiento de un valor con parte fraccionaria requiere que almacenemos no solo el patrón de 0s y 1s correspondiente a su representación binaria, sino también la posición del punto de separación. Una forma muy popular de hacer esto está basada en la notación científica y se conoce como notación en **punto (o coma) flotante**.

Notación en punto flotante

Expliquemos la notación en punto flotante utilizando solo un byte de almacenamiento. Aunque normalmente las máquinas utilizan patrones mucho más largos, este formato de 8 bits es representativo de los sistemas reales y sirve para demostrar los conceptos importantes sin el engorro de emplear largos patrones de bits.

En primer lugar, designamos el bit de mayor peso de ese byte como bit de signo. De nuevo, un 0 en el bit de signo indicará que el valor almacenado es no negativo, mientras que un 1 indicará que el valor es negativo. A continuación, dividimos los restantes 7 bits del byte en dos grupos o campos: el **campo de exponente** y el **campo de mantisa**. Designaremos los 3 bits situados a continuación del bit de signo como el campo de exponente y los restantes 4 bits como el campo de mantisa. La Figura 1.26 ilustra cómo está dividido el byte.

Podemos explicar el significado de los campos considerando el siguiente ejemplo. Suponga un byte compuesto por el patrón de bits 01101011. Analizando este patrón con el formato precedente, vemos que el bit de signo es 0, el exponente es 110 y la mantisa es 1011. Para decodificar el byte, primero extraemos la mantisa y colocamos un punto raíz a su izquierda, obteniendo

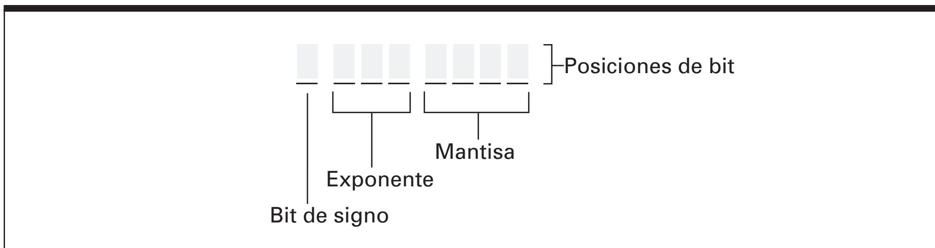
```
.1011
```

A continuación, extraemos el contenido del campo de exponente (110) y lo interpretamos como un entero almacenado utilizando el método de notación en exceso de 3 bits (consulte de nuevo la Figura 1.25). De ese modo, el patrón en el campo de exponente de nuestro ejemplo representa un valor 2 positivo. Esto nos indica que debemos mover el punto raíz de nuestra solución hacia la derecha un total de 2 posiciones. (Un exponente negativo indicaría que hay que mover el punto raíz hacia la izquierda.) En consecuencia, obtenemos

```
10.11
```

que es la representación binaria de $2\frac{3}{4}$. A continuación, observamos que el bit de signo de nuestro ejemplo es 0; el valor representado es por tanto negativo.

Figura 1.26 Componentes de la notación en punto flotante.



Con ello concluimos que el 01101011 representa el valor $2\frac{3}{4}$. Si el patrón hubiera sido 11101011 (que es el mismo que antes excepto por el bit de signo), el valor representado habría sido $-2\frac{3}{4}$.

Veamos otro ejemplo. Considere el byte 00111100. Extraemos la mantisa para obtener

```
.1100
```

y movemos el punto raíz un bit hacia la izquierda, ya que el campo de exponente (011) representa el valor −1. Con ello tenemos

```
.01100
```

que representa el valor $\frac{3}{8}$. Dado que el bit de signo del patrón original es 0, el valor almacenado será positivo. Con ello, concluimos que el patrón 00111100 representa el valor $\frac{3}{8}$.

Para almacenar un valor utilizando notación en punto flotante, invertimos el procedimiento anterior. Por ejemplo, para codificar el valor $1\frac{1}{8}$, primero lo expresamos en notación binaria y obtenemos 1.001. A continuación, copiamos el patrón de bits en el campo de mantisa de izquierda a derecha, comenzando con el 1 situado más a la izquierda en la representación binaria. Llegados a este punto, el byte tendrá el siguiente aspecto:

```
_ _ _ _ 1 0 0 1
```

Ahora deberemos rellenar el campo de exponente. Para ello, nos imaginamos el contenido del campo de mantisa con un punto raíz situado a su izquierda y determinamos el número de bits que habrá que mover el punto raíz, y la dirección en que habrá que desplazarlo para obtener el número binario original. En nuestro ejemplo, vemos que el punto raíz en .1001 tiene que moverse 1 bit hacia la derecha para obtener 1.001. Por tanto, el exponente debe ser un uno positivo, por lo que colocaremos los bits 101 (que es un uno positivo en notación exceso cuatro, como se muestra en la Figura 1.25) en el campo de exponente. Finalmente, rellenamos el bit de signo con el valor 0 porque el valor que estamos almacenando es no negativo. El byte terminado tendrá el siguiente aspecto:

```
0 1 0 1 1 0 0 1
```

Hay un aspecto sutil que es posible que se le haya pasado por alto al rellenar el campo de mantisa. La regla consiste en copiar el patrón de bits que aparece en la representación binaria de izquierda a derecha, comenzando por el 1 situado más a la izquierda. Para clarificar el proceso, vamos a considerar el proceso de almacenar el valor $\frac{3}{8}$, que es .011 en notación binaria. En este caso, la mantisa será

```
_ _ _ _ 1 1 0 0
```

Y no sería

```
_ _ _ _ 0 1 1 0
```

Esto se debe a que debemos rellenar el campo de mantisa *comenzando con el 1 situado más a la izquierda* que aparece en la representación binaria. Las representaciones que cumplen con esta regla se dice que están en **forma normalizada**.

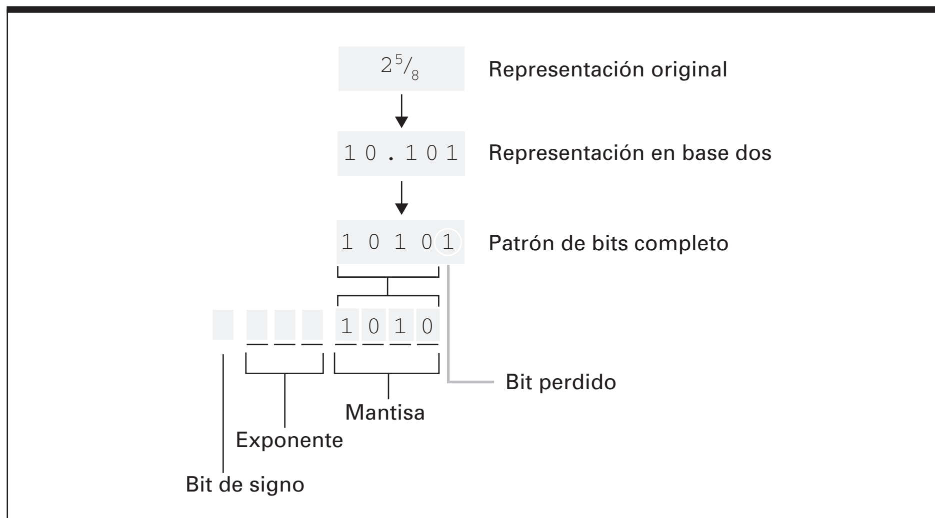
Utilizando la forma normalizada se elimina la posibilidad de que existan múltiples representaciones para un mismo valor. Por ejemplo, tanto 00111100 como 01000110 nos darían, al decodificarlos, el valor $\frac{3}{8}$, pero solo el primero de estos patrones está en forma normalizada. El cumplir con la forma normalizada también implica que la representación para todos los valores distintos de cero tendrá una mantisa que comenzará con 1. Sin embargo, el valor cero es un caso especial; su representación en punto flotante es un patrón de bits compuesto por todo 0s.

Errores de truncamiento

Consideremos el engorroso problema que surge si intentamos almacenar el valor $2\frac{5}{8}$ en nuestro sistema de punto flotante de un solo byte. Escribimos primero $2\frac{5}{8}$ en binario, lo que nos da 10.101. Pero al copiar esto en el campo de mantisa, nos quedamos sin espacio y el 1 situado más a la derecha (que representa el último $\frac{1}{8}$) se pierde (Figura 1.27). Si ignoramos este problema por el momento y continuamos llenando el campo de exponente y el bit de signo, terminamos obteniendo el patrón de bits 01101010, que representa el valor $2\frac{1}{2}$ en lugar de $2\frac{5}{8}$. Lo que ha ocurrido es lo que se conoce como **error de truncamiento** o **error de redondeo**, lo que quiere decir que parte del valor que se está almacenando se pierde debido a que el campo de mantisa no es lo suficientemente largo.

La importancia de estos errores puede reducirse empleando un campo de mantisa más largo. De hecho, la mayor parte de las computadoras que se fabrican hoy día utilizan al menos 32 bits para almacenar valores en notación de punto flotante en lugar de los 8 bits que hemos utilizado aquí. Esto también permite disponer al mismo tiempo de un campo de exponente más largo. Pero incluso con estos formatos más largos seguirá habiendo ocasiones en las que se necesite una mayor precisión.

Figura 1.27 Codificación del valor $2\frac{5}{8}$.



Otra clase de errores de truncamiento es un fenómeno al que ya estamos acostumbrados en la notación en base diez: el problema de las expansiones fraccionarias infinitas, como por ejemplo la que nos encontramos al tratar de expresar $\frac{1}{3}$ en formato decimal. Algunos valores no pueden expresarse con precisión independientemente del número de dígitos que utilicemos. La diferencia entre nuestra notación tradicional en base diez y la notación binaria es que hay más valores con representaciones infinitas en binario que en notación decimal. Por ejemplo, el valor de un décimo tiene una expansión infinita cuando se expresa en binario. Imagine los problemas que esto puede provocar a la persona que no sea consciente de este fenómeno y que esté usando la notación en punto flotante para almacenar y manipular euros y céntimos. En particular, si se utiliza el euro como unidad de medida, el valor de una moneda de diez céntimos no podría almacenarse de forma precisa. Una solución en este caso consiste en manipular los datos en unidades de céntimos, para que todos los valores sean enteros y puedan almacenarse de forma precisa utilizando un método tal como el complemento a dos.

Los errores de truncamiento y sus problemas relacionados constituyen una preocupación constante para las personas que trabajan en el área del análisis numérico. Esta rama de las matemáticas se ocupa de los problemas que surgen al realizar cálculos reales que a menudo son masivos y que requieren una precisión suficiente.

A continuación proporcionamos un ejemplo que suscitaría el interés de cualquier analista numérico. Suponga que nos piden sumar los tres valores siguientes empleando nuestra notación en punto flotante de un solo byte que hemos definido anteriormente:

$$2\frac{1}{2} + \frac{1}{8} + \frac{1}{8}$$

Punto flotante de simple precisión

La notación en punto flotante que hemos presentado en este capítulo (Sección 1.7) es demasiado simplista como para poder emplearla en una computadora real. Después de todo, con solo 8 bits únicamente podemos expresar 256 números de entre el conjunto de todos los números reales. En nuestras explicaciones hemos utilizado 8 bits para hacer que los ejemplos fueran simples, lo que no impedía resaltar la importancia de los conceptos subyacentes.

Muchas de las computadoras actuales soportan una variante de 32 bits de esta notación denominada **punto flotante de simple precisión**. Este formato utiliza 1 bit para el signo, 8 bits para el exponente (con notación en exceso) y 23 bits para la mantisa. Por tanto, la notación en punto flotante de simple precisión es capaz de expresar números muy grandes (del orden de 10^{38}) y también números muy pequeños (del orden de 10^{-37}) con una precisión de 7 dígitos decimales. Es decir, los primeros 7 dígitos de un número decimal dado pueden almacenarse con una muy buena precisión (aunque de todos modos puede existir un pequeño error). Cualquier dígito situado más allá de los 7 primeros se perderá, casi con toda seguridad, debido a los errores de truncamiento (aunque el módulo del número se conserva). Otro formato denominado **punto flotante de doble precisión** utiliza 64 bits y proporciona una precisión de 15 dígitos decimales.

Si sumamos los valores en el orden indicado, sumaremos primero $2\frac{1}{2}$ y $\frac{1}{8}$ para obtener $2\frac{5}{8}$, que en binario es 10.101. Desafortunadamente, dado que este valor no puede almacenarse con precisión (como hemos visto anteriormente), el resultado de nuestro primer paso termina siendo almacenado como $2\frac{1}{2}$ (que es igual a uno de los valores que estamos sumando). El siguiente paso consiste en sumar este resultado al último valor $\frac{1}{8}$. De nuevo, nos encontramos con el mismo error de truncamiento y el resultado final resulta ser la respuesta incorrecta $2\frac{1}{2}$.

Sumemos ahora los valores en el orden opuesto. Primero sumamos $\frac{1}{8}$ y $\frac{1}{8}$ para obtener $\frac{1}{4}$. En binario, este resultado es .01; así que el resultado de nuestro primer paso se almacena en un byte como 00111000, que es una representación precisa. Ahora sumamos este $\frac{1}{4}$ al siguiente valor de la lista, $2\frac{1}{2}$, y obtenemos $2\frac{3}{4}$, que podemos almacenar con total precisión en un byte como 01101011. El resultado en esta ocasión es la respuesta correcta.

En resumen, al sumar valores numéricos representados en punto flotante, el orden en que se sumen puede ser importante. El problema es que si sumamos un número muy grande con otro muy pequeño, el número pequeño puede verse truncado. Por tanto, la regla general para sumar varios valores consiste en sumar primero los valores pequeños, esperando que se acumulen para dar un valor que sea significativo a la hora de sumarlo con los valores más grandes. Este es precisamente el fenómeno que hemos experimentado en el ejemplo anterior.

Los diseñadores de paquetes software comerciales actuales realizan un excelente trabajo a la hora de proteger al usuario no experto frente a este tipo de problemas. En un sistema de hoja de cálculo típico obtendremos la respuesta correcta a menos que estemos utilizando valores cuyo módulo difiera en un factor de 10^{16} o más. Por tanto, si resulta que necesitamos sumar 1 al valor

```
10,000,000,000,000,000
```

podemos obtener la respuesta

```
10,000,000,000,000,000
```

en lugar de

```
10,000,000,000,000,001
```

Dichos problemas cobran importancia en aplicaciones, como los sistemas de navegación, en las que los pequeños errores pueden acumularse en los cálculos adicionales y llegar a producir consecuencias significativas, pero para el usuario de PC típico, el grado de precisión ofrecido por la mayor parte del software comercial es suficiente.

Cuestiones y ejercicios

1. Decodifique los siguientes patrones de bits utilizando el formato en punto flotante explicado en el texto:

   a. `01001010` **b.** `01101101` **c.** `00111001` **d.** `11011100` **e.** `10101011`

2. Codifique los siguiente valores en el formato de punto flotante explicado en el texto. Indique si se producen errores de truncamiento.

   a. $2\frac{3}{4}$ **b.** $5\frac{1}{4}$ **c.** $\frac{3}{4}$ **d.** $-3\frac{1}{2}$ **e.** $-4\frac{3}{8}$

3. En términos del formato de punto flotante explicado en el texto, ¿cuál de los dos patrones 01001001 y 00111101 representa el valor mayor? Describa un procedimiento simple para determinar cuál de dos patrones cualesquiera representa el valor mayor.
4. Cuando utilizamos el formato de punto flotante explicado en el texto, ¿cuál es el mayor valor que se puede representar? ¿Cuál es el valor positivo más pequeño que se puede representar?

1.8 Compresión de datos

Con el objetivo de almacenar o transferir datos a menudo es útil (y a veces obligatorio) reducir el tamaño de los datos que hay que manipular, al mismo tiempo que se conserva la información subyacente. La técnica para hacer esto se denomina **compresión de datos**. Comenzaremos esta sección teniendo en cuenta algunos métodos genéricos de compresión de datos y luego examinaremos determinadas técnicas diseñadas para aplicaciones específicas.

Técnicas genéricas de compresión de datos

Los esquemas de compresión de datos pueden clasificarse en dos categorías. Algunos de ellos son **sin pérdidas**, mientras que otros son **con pérdidas**. Los esquemas sin pérdidas son aquellos en los que no se pierde información durante el proceso de compresión. Los esquemas con pérdidas son esos otros que pueden llevar a la pérdida de una parte de la información. Las técnicas con pérdidas proporcionan a menudo un mayor grado de compresión que los que no tienen pérdidas y son, por tanto, bastante populares en aquellos entornos en los que pueden tolerarse los errores poco significativos, como es el caso de las imágenes y del audio.

En aquellos casos en los que los datos que se están comprimiendo están formados por largas secuencias del mismo valor, una técnica de compresión muy popular es la denominada **codificación por longitud de secuencia**, que es un método de compresión sin pérdidas. Dicha técnica es el proceso de sustituir las secuencias de elementos de datos con un código que indica el elemento repetido y el número de veces que ese elemento aparece dentro de la secuencia. Por ejemplo, esto permite reducir el espacio requerido para indicar que un patrón de bits está compuesto por 253 unos, seguidos de 118 ceros y seguidos de 87 unos. El número de bits almacenados con esta técnica de compresión será mucho menor que si enumeráramos los 458 bits que componen el patrón.

Otra técnica de compresión de datos sin pérdidas es la **codificación dependiente de la frecuencia**, un sistema en el que la longitud del patrón de bits utilizado para representar un elemento de datos es inversamente propor-

cional a la frecuencia con que ese elemento aparece. Dichos códigos son un ejemplo de los denominados códigos de longitud variable, que son aquellos en los que los elementos se representan mediante patrones de diferentes longitudes, por contraste con otros códigos tales como Unicode, en los que todos los símbolos se representan mediante 16 bits. Se atribuye a David Huffman el descubrimiento de un algoritmo que se utiliza de forma bastante común para el desarrollo de códigos dependientes de la frecuencia, por lo que los códigos desarrollados de esta manera suelen referirse con el nombre de **códigos de Huffman**. A su vez, la mayoría de los códigos dependientes de la frecuencia que se emplean actualmente son códigos de Huffman.

Como ejemplo de codificación dependiente de la frecuencia, consideremos la tarea de codificar un texto en inglés. En el idioma inglés, las letras *e, t, a,* e *i* se utilizan con más frecuencia que las letras *z, q* y *x.* Por tanto, a la hora de construir un código para un texto escrito en inglés, podemos ahorrar espacio utilizando patrones de bits cortos para representar el primer conjunto de letras y patrones más largos para representar el otro conjunto. El resultado será un código en el que cualquier texto escrito en inglés tendrá una representación más corta que la que se obtendría utilizando códigos de longitud uniforme.

En algunos casos, el flujo de datos que hay que comprimir está compuesto por unidades, cada una de las cuales solo difiere ligeramente de la anterior. Un ejemplo sería las imágenes consecutivas de una película. En estos casos, es útil emplear técnicas de **codificación relativa**, también conocidas como **codificación diferencial**. Estas técnicas almacenan las diferencias entre unidades de datos consecutivas en lugar de almacenar las unidades completas; es decir, cada unidad se codifica en términos de su relación con la unidad anterior. La codificación relativa puede implementarse mediante técnicas de compresión sin pérdidas o con pérdidas, dependiendo de si las diferencias entre unidades de datos consecutivas están codificadas de forma precisa o aproximada.

Otro conjunto bastante popular de sistemas de compresión está basado en las técnicas de **codificación por diccionario**. En este caso, el término *diccionario* hace referencia a una colección de bloques componentes a partir de los cuales se construye el mensaje que se quiere comprimir, codificándose el propio mensaje mediante una serie de referencias al propio diccionario. Normalmente, consideramos los sistema de codificación por diccionario como sistemas sin pérdidas, pero como veremos en nuestro análisis de las técnicas de compresión de imágenes, hay veces en las que las entradas del diccionario son solo aproximaciones de los elementos de datos correctos, por lo que se obtiene un sistema de compresión con pérdidas.

La codificación por diccionario puede ser utilizada por los procesadores de texto para comprimir documentos de texto, porque los diccionarios ya integrados en dichos procesadores para la corrección ortográfica proporcionan excelentes diccionarios de compresión. En particular, podemos codificar una palabra completa mediante una única referencia a dicho diccionario, en lugar de como una secuencia de caracteres individuales codificados utilizando un sistema tal como ASCII o Unicode. Un diccionario típico en un procesador de textos contiene aproximadamente 25.000 entradas, lo que quiere decir que cada entrada individual puede identificarse mediante un entero comprendido en el rango que va desde 0 a 24.999. Esto significa que cada entrada concreta del diccionario puede identificarse mediante un patrón de solo 15 bits. Por

contraste, si la palabra a la que se quiere hacer referencia está formada por seis letras, su codificación carácter a carácter requeriría 48 bits utilizando ASCII de 8 bits o 96 bits utilizando Unicode.

Una variante de la técnica de codificación por diccionario es la **codificación por diccionario adaptativa** (también denominada codificación por diccionario dinámica). En un sistema de codificación por diccionario adaptativa, se permite que el diccionario varíe durante el proceso de codificación. Un ejemplo muy popular es la **codificación de Lempel-Ziv-Welsh (LZW)** (llamada así en honor de sus creadores, Abraham Lempel, Jacob Ziv y Terry Welsh). Para codificar un mensaje utilizando LZW, comenzamos con un diccionario que contiene los bloques componentes básicos a partir de los cuales se construye el mensaje, pero a medida que vamos encontrando unidades de mayor tamaño en el mensaje, se añaden esas unidades al diccionario, lo que implica que las futuras apariciones de esas unidades pueden codificarse mediante una única referencia al diccionario en lugar de mediante múltiples referencias. Por ejemplo, a la hora de codificar un texto escrito en inglés, podríamos comenzar con un diccionario que contuviera los caracteres individuales, los dígitos y los signos de puntuación. Pero a medida que vamos identificando palabras en el mensaje, se pueden ir añadiendo esas palabras al diccionario. De este modo, el diccionario crecería a medida que se codifica el mensaje y, según vaya creciendo el diccionario más palabras del mensaje (o patrones recurrentes de palabras) podremos codificar mediante una única referencia al diccionario.

El resultado será un mensaje codificado en términos de un diccionario más bien grande, que será único para ese mensaje concreto. Pero este diccionario de gran tamaño no es necesario para decodificar el mensaje. Lo único que hace falta es disponer del pequeño diccionario original. Efectivamente, el proceso de decodificación comenzaría con el mismo pequeño diccionario con el que comenzó el proceso de codificación. Después, a medida que se desarrolla el proceso de decodificación, irá encontrando las mismas unidades que se hayan encontrado durante el proceso de codificación, por lo que se las podrá ir añadiendo al diccionario para futura referencia exactamente igual que se hizo anteriormente al codificar el mensaje.

Para clarificar esta técnica, vamos a ver qué pasa cuando se aplica la codificación LZW al mensaje

xyx xyx xyx xyx

partiendo de un diccionario en el que solo hay tres entradas, siendo la primera de ellas *x*, la segunda *y* y la tercera el carácter espacio. Comenzaríamos codificando *xyx* mediante el valor 121, lo que quiere decir que el mensaje comienza con un patrón compuesto por la primera entrada del diccionario, seguida de la segunda y seguida de la primera. A continuación, codificamos el espacio para obtener el patrón 1213. Pero, habiendo llegado a un espacio, sabemos que la cadena anterior de caracteres forma una palabra, por lo que añadimos el patrón *xyx* al diccionario como cuarta entrada. Continuando de esta manera, codificaríamos todo el mensaje mediante el patrón 121343434.

Si ahora se nos pidiera decodificar este mensaje partiendo del diccionario original de tres entradas, comenzaríamos decodificando la cadena inicial 1213 como *xyx* seguido de un espacio. En este punto, nos daríamos cuenta de que la

cadena *xyx* forma una palabra y la añadiríamos al diccionario como cuarta entrada, exactamente igual que hicimos durante el proceso de codificación. Después, continuaríamos decodificando el mensaje, dándonos cuenta de que el 4 contenido en el mensaje hace referencia a esta cuarta entrada y decoficándolo como las palabra *xyx*, con lo que obtendríamos el patrón

xyx xyx

Continuando de esta forma terminaríamos decodificando la cadena 121343434 como

xyx xyx xyx xyx

que es el mensaje original.

Compresión de imágenes

En la Sección 1.4, hemos visto cómo se codifican las imágenes utilizando técnicas de mapa de bits. Lamentablemente, los mapas de bits producidos suelen ser de muy gran tamaño. Por ello, se han desarrollado numerosos esquemas de compresión específicos para representar imágenes.

Un sistema, conocido con el nombre de **GIF** (*Graphic Interchange Format*, Formato de intercambio de gráficos) es un sistema de codificación por diccionario que fue desarrollado por CompuServe. Este sistema trata de resolver el problema de la compresión reduciendo a solo 256 el número de colores que pueden asignarse a un píxel. La combinación rojo-verde-azul (red-green-blue) para cada uno de estos colores se codifica utilizando tres bytes y estas 256 codificaciones se almacenan en una tabla (un diccionario) que se denomina paleta. Después, cada píxel de una imagen puede representarse mediante un único byte, cuyo valor indica cuál de las 256 entradas de la paleta representa el color del píxel. (Recuerde que un único byte puede contener uno cualquiera de 256 patrones de bits diferentes.) Observe que GIF es un sistema de compresión con pérdidas cuando lo aplicamos a imágenes arbitrarias, porque los colores de la paleta pueden no ser iguales a los colores de la imagen original.

El sistema GIF puede incrementar el grado de compresión ampliando este sistema de diccionario simple a un sistema de diccionario adaptativo utilizando técnicas LZW. En particular, a medida que se encuentran patrones de píxeles durante el proceso de codificación, dichos patrones se añaden al diccionario para poder codificar de forma más eficiente las futuras apariciones de esos patrones. De ese modo, el diccionario final está compuesto por la paleta original y una colección de patrones de píxeles.

A uno de los colores de la paleta GIF se le asigna normalmente el valor “transparente”, lo que quiere decir que se permite que se vea el fondo a través de todas las regiones a las que se les haya asignado dicho “color”. Esta opción, combinada con la relativa simplicidad del sistema GIF, hace que GIF sea la opción preferida en las aplicaciones simples de animación, en las que es preciso desplazar múltiples imágenes por la pantalla de la computadora. Por otro lado, su capacidad de codificar únicamente 256 colores hace que este sistema sea poco adecuado para aquellas aplicaciones en las que hace falta una mayor precisión, como sucede en el campo de la fotografía.

Otro sistema de compresión muy popular es el sistema **JPEG**. Es un estándar desarrollado por el **Joint Photographic Experts Group** (Grupo conjunto

de expertos en fotografía), de aquí el nombre del estándar dentro de la organización ISO. JPEG ha demostrado ser un estándar muy eficiente para la compresión de fotografías en color y se utiliza ampliamente en el sector fotográfico, como lo indica el hecho de que la mayoría de las cámaras digitales emplean JPEG como técnica predeterminada de compresión.

En realidad, el estándar JPEG abarca diversos métodos de compresión de imágenes, cada uno con sus propios objetivos. En aquellas situaciones en las que se requiere una precisión máxima, JPEG proporciona un modo de compresión sin pérdidas. Sin embargo, el modo JPEG sin pérdidas no permite obtener un alto grado de compresión si lo comparamos con otras opciones JPEG. Además, hay otras opciones JPEG que han conseguido mucha mayor aceptación, por lo que el modo JPEG sin pérdidas raramente se emplea. En lugar de ello, el estándar preferido en muchas aplicaciones es la opción denominada estándar de línea base JPEG (también conocida con el nombre de modo secuencial con pérdidas de JPEG).

La compresión de imágenes utilizando el estándar de línea base JPEG requiere una secuencia de pasos, algunos de los cuales están diseñados para aprovecharse de las limitaciones del ojo humano. En particular, el ojo humano es más sensible a los cambios en el brillo que a los cambios de color. Por eso, si partimos de una imagen que esté codificada en función de sus componentes de luminancia y crominancia, el primer paso consiste en promediar los valores de crominancia en cuadrados de dos por dos píxeles. Esto reduce el tamaño de información de crominancia en un factor de cuatro, mientras que se conserva toda la información original de brillo. El resultado es un grado de compresión significativo sin una pérdida apreciable de calidad de la imagen.

El siguiente paso consiste en dividir la imagen en bloques de ocho por ocho píxeles y comprimir la información de cada bloque como una sola unidad. Esto se hace aplicando una técnica matemática conocida con el nombre de transformada discreta de coseno, de cuyos detalles no nos tenemos que preocupar aquí. Lo importante es que dicha transformación convierte el bloque original de ocho por ocho píxeles en otro bloque, cuyas entradas reflejan el modo en que los píxeles del bloque original se relacionan entre sí en lugar de reflejar los valores reales de cada píxel. En este nuevo bloque, se sustituyen por cero los valores que se encuentran por debajo de un umbral predeterminado, para reflejar el hecho de que los cambios representados por dichos valores son demasiado sutiles como para poder ser detectados por el ojo humano. Por ejemplo, si el bloque original contiene un patrón ajedrezado, el nuevo bloque podría reflejar la existencia de un color promedio uniforme. (Un bloque típico de ocho por ocho píxeles representaría un cuadrado muy pequeño dentro de la imagen, por lo que el ojo humano sería incapaz, de todos modos, de identificar la apariencia ajedrezada.)

Llegados a este punto, se aplican técnicas más tradicionales de codificación por longitud de secuencia, de codificación relativa y de codificación de longitud variable para obtener un grado de compresión adicional. En conjunto, el estándar de base de línea JPEG suele comprimir las imágenes en color según un factor de al menos 10, y a veces se puede alcanzar un factor de 30, sin una pérdida apreciable de calidad.

Otro sistema más de compresión de datos asociado con el tratamiento de imágenes es **TIFF** (*Tagged Image File Format*, Formato etiquetado de archivos

de imagen). Sin embargo, el uso más popular de TIFF no es como medio de comprimir los datos sino más bien como formato estandarizado para el almacenamiento de fotografías junto con información relacionada con ellas como la fecha, la hora y las opciones de la cámara. En este contexto, la propia imagen suele almacenarse mediante sus componentes roja, verde y azul de cada píxel, sin utilizar compresión.

El conjunto de estándares TIFF sí que incluye técnicas de compresión de datos, la mayoría de las cuales están diseñadas para comprimir imágenes de documentos de texto en aplicaciones de facsímil. Dichas técnicas emplean variantes de la codificación por longitud de secuencia para aprovechar el hecho de que los documentos de texto están compuestos por largas cadenas de píxeles en blanco. La opción de compresión de imágenes en color incluida en los estándares TIFF está basada en técnicas similares a las utilizadas por GIF, por lo que se emplea ampliamente en el sector de la fotografía.

Compresión de audio y vídeo

Los estándares más comúnmente utilizados para la codificación y compresión de audio y vídeo fueron desarrollados por **MPEG** (***Motion Picture Experts Group***, Grupo de expertos en imágenes en movimiento) bajo la dirección de ISO. Por ello, estos estándares se suelen denominar MPEG.

MPEG abarca diversos estándares para diferentes aplicaciones. Por ejemplo, las necesidades de la difusión de televisión de alta definición (HDTV, ***high definition television***) son diferentes de las de una videoconferencia, en las que la señal emitida debe buscar su camino a través de diversas rutas de comunicaciones que pueden tener capacidades limitadas. Y las necesidades de estas dos aplicaciones difieren, a su vez, de las necesidades que surgen al intentar almacenar vídeo de forma y manera que puedan reproducirse o saltarse las diversas secciones del mismo.

Las técnicas empleadas por MPEG caen fuera del alcance de este libro, pero en general las técnicas de compresión de vídeo están basadas en el concepto de que un vídeo se construye como una secuencia de imágenes, de forma muy similar a como se graban en una película las imágenes de un éxito cinematográfico. Para comprimir tales secuencias, solo se codifican completamente algunas de las imágenes, a las que se denomina imágenes I. Las imágenes comprendidas entre dos imágenes I sucesivas se codifican empleando técnicas de codificación relativa. Es decir, en lugar de codificar la imagen completa, solo se graban las variaciones con respecto a la imagen anterior. Las propias imágenes I se suelen comprimir utilizando técnicas similares a JPEG.

El sistema más conocido para la compresión de audio es **MP3**, que fue desarrollado como parte de los estándares MPEG. De hecho, el acrónimo *MP3* es la abreviatura de *MPEG capa 3.* Entre otras técnicas de compresión, MP3 aprovecha las características del oído humano, eliminando todos los detalles que este no puede percibir. Una de esas propiedades, denominada **enmascaramiento temporal**, es que durante un corto periodo de tiempo después de percibirse un sonido de gran intensidad, el oído humano no puede detectar otros sonidos más débiles, que en condiciones normales sí que serían audibles. Otra característica, denominada **enmascaramiento de frecuencia**, es que un sonido de una determinada frecuencia tiende a enmascarar los sonidos más

débiles emitidos con frecuencias próximas a esa. Aprovechando tales características, MP3 puede utilizarse para conseguir un grado significativo de compresión del audio, al mismo tiempo que se mantiene una calidad de sonido similar a la que puede conseguirse con un CD.

Utilizando las técnicas de compresión MPEG y MP3, las videocámaras son capaces de grabar hasta una hora de vídeo en 128MB de almacenamiento, mientras que los reproductores portátiles de música pueden almacenar hasta 400 canciones en un único GB. Pero, a diferencia de los objetivos de las técnicas de compresión en otros entornos, el objetivo al comprimir audio y vídeo no es necesariamente ahorrar espacio de almacenamiento. Hay otro objetivo igualmente importante, que es el de obtener una codificación que permita transmitir información a través de los actuales sistemas de comunicaciones con la suficiente rapidez como para poder presentar esa información a tiempo. Si cada trama de vídeo requiriera un MB de almacenamiento y hubiera que transmitir las tramas a través de una ruta de comunicaciones que solo pueda retransmitir un KB por segundo, no podría nunca mantenerse una videoconferencia. Por tanto, además de la calidad permitida de la reproducción, los sistemas de compresión de audio y vídeo suelen juzgarse según la velocidad de transmisión requerida para la comunicación temporizada de los datos. Estas velocidades normalmente se miden en **bits por segundo (bps)**. Las unidades más comunes son los **Kbps** (kilo-bps, que equivale a mil bps), **Mbps** (mega-bps, que equivale a un millón de bps) y **Gbps** (giga-bps, igual a mil millones de bps). Utilizando técnicas MPEG, pueden retransmitirse presentaciones de vídeo a través de rutas de comunicaciones que proporcionan tasas de transferencia de 40 Mbps. Generalmente, las grabaciones MP3 suelen requerir tasas de transferencia no superiores a 64 Kbps.

Cuestiones y ejercicios

1. Enumere cuatro técnicas genéricas de compresión.
2. ¿Cuál sería la versión codificada del mensaje

   ```
   xyx yxxxy xyx yxxxy yxxxy
   ```

   si utilizamos compresión LZW, partiendo del diccionario que contiene los símbolos *x*, *y* y el espacio (como se describe en el texto)?
3. ¿Por qué GIF es más adecuado que JPEG para codificar dibujos animados en color?
4. Suponga que forma parte de un equipo que está diseñando una nave espacial que va a viajar a otros planetas y a enviar de vuelta imágenes fotográficas. ¿Sería buena idea comprimir las fotografías usando GIF o el estándar de línea base JPEG con el fin de reducir los recursos requeridos para almacenar y transmitir las imágenes?
5. ¿Qué característica del ojo humano aprovecha el estándar de línea base JPEG?
6. ¿Qué característica del oído humano aprovecha el estándar MP3?
7. Indique un fenómeno problemático que es común a la hora de codificar información numérica, imágenes y sonidos en forma de patrones de bits.

1.9 Errores de comunicación

Cuando se intercambia información entre distintas partes de una computadora, o se transmite información desde la Tierra a la Luna, y viceversa, o también cuando se almacena simplemente la información, siempre existe la posibilidad de que el patrón de bits extraído al final no sea idéntico al patrón original. Las partículas de polvo o de grasa en la superficie magnética de grabación o los problemas de funcionamiento de un circuito pueden hacer que los datos se graben o se lean de manera incorrecta. El ruido estático en una ruta de transmisión puede corromper determinadas partes de los datos y, en el caso de algunas tecnologías, la radiación de fondo normal puede alterar los patrones almacenados en la memoria principal de la máquina.

Para resolver tales problemas, se han desarrollado diversas técnicas de codificación que permiten la detección e incluso la corrección de errores. Hoy día, dado que esas técnicas están integradas en buena medida en los componentes internos de un sistema de computadoras, las personas que utilizan la máquina no suelen percatarse de su existencia. Y sin embargo, su presencia es importante y representa una contribución significativa a la investigación científica. Es conveniente, por tanto, que investiguemos algunas de estas técnicas que permiten alcanzar el alto grado de fiabilidad que ofrecen los equipos actuales.

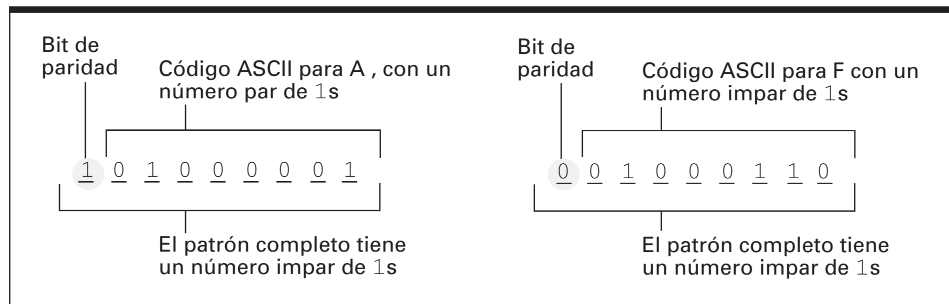
Bits de paridad

Un método simple para la detección de errores está basado en el principio de que si cada patrón de bits que está siendo manipulado tiene un número impar de 1s y nos encontramos con un patrón con un número par de 1s, podemos deducir que se ha producido algún tipo de error. Para utilizar este principio, necesitamos un sistema de codificación en el que cada patrón contenga un número impar de 1s. Esto puede conseguirse fácilmente añadiendo primero un bit adicional, denominado **bit de paridad**, a cada patrón de un determinado sistema de codificación ya disponible (por ejemplo, se puede añadir el bit de paridad en el extremo de mayor peso). En cada caso, asignamos el valor 1 o 0 a este nuevo bit, de modo que el patrón completo resultante tenga un número impar de 1s. Una vez modificado nuestro sistema de codificación de esta forma, el encontrarnos con un patrón con un número par de 1s indicará que se ha producido algún error y que el patrón que estamos manipulando es incorrecto.

En la Figura 1.28 se muestra cómo se podrían añadir bits de paridad a los códigos ASCII correspondientes a las letras A y F. Observe que el código para A pasa a ser 101000001 (bit de paridad 1) y que el de la letra F se convierte en 001000110 (bit de paridad igual 0). Aunque el patrón original de 8 bits para A tiene un número par de 1s y el patrón original de 8 bits de F tiene un número impar de 1s, los dos patrones de 9 bits resultantes tienen un número impar de 1s. Si aplicáramos esta técnica a todos los patrones ASCII de 8 bits, obtendríamos un sistema de codificación de 9 bits en el que el encontrar un patrón de 9 bits con un número par de 1s indicaría sin ningún género de duda que se ha producido un error.

El sistema de paridad que acabamos de describir se denomina sistema de **paridad impar**, porque hemos diseñado nuestro sistema de modo que cada patrón correcto contenga un número impar de 1s. Otra técnica alternativa es

Figura 1.28 Códigos ASCII de las letras A y F ajustados para paridad impar.



la que se denomina de **paridad par**. En un sistema de paridad par se diseña cada patrón para que contenga un número par de 1s, de manera que la presencia de errores estará indicada por la aparición de un patrón con un número impar de 1s.

Hoy día, es bastante usual que se empleen bits de paridad en la memoria principal de las computadoras. Aunque contemplamos estas máquinas como si tuvieran celdas de memoria con una capacidad de 8 bits, en realidad cada celda tiene una capacidad de 9 bits, empleando uno de esos bits como bit de paridad. Cada vez que se entrega un patrón de 8 bits a la circuitería de memoria para su almacenamiento, esa circuitería añade un bit de paridad y almacena el patrón de 9 bits resultante. Cuando el patrón se extrae posteriormente, la circuitería comprueba la paridad del patrón de 9 bits. Si no se detecta ningún error, la memoria elimina el bit de paridad y devuelve con toda tranquilidad el patrón de 8 bits restante. En caso contrario, la memoria devuelve los 8 bits de datos, pero con una advertencia de que el patrón que se devuelve puede no ser el mismo que el que originalmente se almacenó en la memoria.

Esta utilización tan sencilla y directa de los bits de paridad es simple, pero tiene sus limitaciones. Si un patrón tiene un número impar de 1s y se produjeran dos errores, el patrón seguirá teniendo un número impar de 1s, por lo que el sistema de paridad no detectará los errores que se han producido. De hecho, esta aplicación tan sencilla de los bits de paridad no puede detectar ningún conjunto de errores dentro de un patrón compuesto por un número par de errores.

En ocasiones, a los patrones de bits de gran longitud, como por ejemplo la cadena de bits que se graba en un sector de un disco magnético, se les aplica un método para minimizar este problema relativo a los errores no detectados. Lo que se hace en dicho caso es acompañar el patrón por un conjunto de bits de paridad que forman lo que se denomina **byte de comprobación**. Cada bit del byte de comprobación es un bit de paridad asociado con un conjunto concreto de bits dispersos a lo largo de todo el patrón. Por ejemplo, podría asociarse un bit de paridad con el bit inicial de cada byte que componga el patrón, mientras que otro bit de paridad podría estar asociado con el segundo bit de cada byte que forman el patrón. De esta forma, resulta bastante probable que se pueda detectar un conjunto de errores concentrado en el área del patrón original, ya que afectará a varios de los bits de paridad. Una serie de variantes de este concepto de los bytes de comprobación condujo hace ya tiempo al desarrollo de

esquemas de detección de errores conocidos como **sumas de comprobación** y **códigos de redundancia cíclica (CRC)**.

Códigos de corrección de errores

Aunque el uso de un bit de paridad permite detectar un error, no proporciona la información necesaria para corregir el error. Hay muchas personas que se sorprenden al saber que se pueden diseñar **códigos de corrección de errores** que no solo permiten detectar los errores, sino también corregirlos. No es extraño que esas personas se sorprendan: después de todo, la intuición nos dice que no deberíamos poder corregir los errores de un mensaje recibido a menos que conozcamos ya la información original contenida en el mensaje. Sin embargo, en la Figura 1.29 se muestra un código simple que sí que dispone de dicha propiedad de corrección.

Para comprender cómo funciona este código, vamos a definir el concepto de **distancia de Hamming**, que debe su nombre a R. W. Hamming, que fue el pionero en la búsqueda de códigos de corrección de errores, después de la frustración que le causó la falta de fiabilidad de las primeras máquinas basadas en relés de la década de 1940. La distancia de Hamming entre dos patrones de bits es el número de bits en que difieren ambos patrones. Por ejemplo, la distancia de Hamming entre los patrones que representan a las letras A y B en el código de la Figura 1.29 es cuatro, mientras que la distancia de Hamming entre B y C es tres. La característica importante del código de la Figura 1.29 es que cualesquiera dos patrones están separados entre sí por una distancia de Hamming de al menos tres.

Si modificamos un único bit en uno cualquiera de los patrones de la Figura 1.29, el error podrá ser detectado ya que el resultado no será un patrón correcto. (Debemos cambiar al menos 3 bits en cualquier patrón para que pase a convertirse en otro patrón correcto.) Además, es perfectamente posible adivinar cuál era el patrón original. Eso se debe a que el patrón modificado tendrá una distancia de Hamming de solo uno con respecto al patrón original, mientras que su distancia con respecto a todos los demás patrones correctos será de al menos dos.

Por tanto, para decodificar un mensaje que haya sido codificado originalmente utilizando la Figura 1.29, no tenemos más que comparar el patrón reci-

Figura 1.29 Código de corrección de errores.

Símbolo	Código
A	000000
B	001111
C	010011
D	011100
E	100110
F	101001
G	110101
H	111010

Figura 1.30 Decodificación del patrón 010100 utilizando el código de la Figura 1.29.

Carácter	Código	Patrón recibido	Distancia entre el patrón recibido y el código
A	0 0 0 0 0 0	0 **1** 0 **1** 0 0	2
B	0 0 1 1 1 1	0 **1** **0** 1 **0** **0**	4
C	0 1 0 0 1 1	0 1 0 **1** **0** **0**	3
D	0 1 1 1 0 0	0 1 **0** 1 0 0	**1** — Distancia más pequeña
E	1 0 0 1 1 0	**0** **1** 0 1 **0** 0	3
F	1 0 1 0 0 1	**0** **1** **0** **1** 0 **0**	5
G	1 1 0 1 0 1	**0** 1 0 1 0 **0**	2
H	1 1 1 0 1 0	**0** 1 **0** **1** **0** 0	4

bido con los patrones del código hasta encontrar uno que esté a una distancia máxima de un bit del patrón recibido. Entonces, consideraremos que ese será el símbolo correcto para la decodificación. Por ejemplo, si recibiéramos el patrón de bits 010100 y lo comparáramos con todos los patrones del código, obtendríamos la tabla de la Figura 1.30. Con eso, podríamos concluir que el carácter transmitido debe haber sido una D, porque es el patrón que más se aproxima al recibido.

Como puede ver, la utilización de esta técnica con el código de la Figura 1.29 nos permite detectar, en la práctica, hasta dos errores por patrón y también nos permite corregir un máximo de un error. Si diseñáramos el código para que el patrón tuviera una distancia de Hamming de al menos cinco con respecto a todos los demás patrones correctos, entonces podríamos detectar hasta cuatro errores por patrón y corregir un máximo de dos. Por supuesto, el diseño de códigos suficientes asociados con grandes distancias de Hamming no es una tarea sencilla. De hecho, constituye una parte de la rama de las matemáticas conocida como teoría de codificación algebraica, que es un tema que cae dentro de los campos del álgebra lineal y de la teoría de matrices.

Las técnicas de corrección de errores se utilizan ampliamente para incrementar la fiabilidad de los equipos de computación. Por ejemplo, se emplean en unidades de disco magnéticas de alta capacidad con el fin de reducir la posibilidad de que los pequeños fallos de la superficie magnética corrompan los datos. Asimismo, una de las principales diferencias entre el formato de CD original utilizado en los discos de audio y el formato posterior empleado para el almacenamiento de datos de computadoras afecta al grado de corrección de errores implicado. El formato de los CD-DA incorpora mecanismos de corrección de errores que reducen la tasa de errores hasta el nivel de un único error cada dos CD. Esto resulta bastante adecuado para grabaciones de audio, pero una empresa que utilice discos CD para suministrar software a los clientes encontraría errores en la mitad de los discos que se grabaran, lo que sería completamente intolerable. Por tanto, en los CD utilizados para el almacenamiento de datos se emplean técnicas adicionales de corrección de errores, reduciendo la probabilidad de error a un único error por cada 20.000 discos.

Cuestiones y ejercicios

1. Los siguientes bytes fueron originalmente codificados utilizando paridad impar. ¿En cuáles de ellos podemos deducir que se ha producido un error?

   a. `100101101` **b.** `100000001` **c.** `000000000`
   d. `111000000` **e.** `011111111`

2. ¿Podrían haberse producido errores en alguno de los bytes de la Cuestión 1 sin que lleguemos a detectarlos? Explique su respuesta.
3. ¿Cómo cambiaría su respuesta a las Cuestiones 1 y 2 si nos dijeran que lo que se ha utilizado es paridad par en lugar de paridad impar?
4. Codifique estas frases en ASCII utilizando paridad impar, añadiendo un bit de paridad en el extremo de mayor peso de cada código de carácter:

   a. "Stop!" Cheryl shouted. **b.** Does 2 + 3 = 5?

5. Utilizando el código de corrección de errores presentado en la Figura 1.29, decodifique los siguientes mensajes:

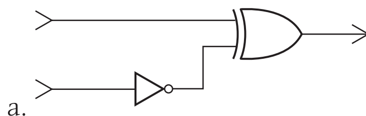
   a. `001111 100100 001100` **b.** `010001 000000 001011`
   c. `011010 110110 100000 011100`

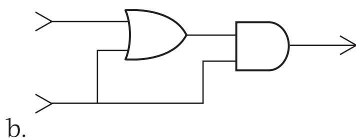
6. Construya un código para los caracteres A, B, C y D utilizando patrones de bits de longitud cinco, de modo que la distancia de Hamming entre cualesquiera dos patrones sea al menos igual a tres.

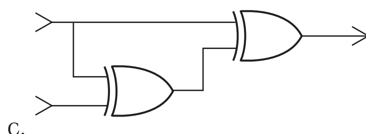
Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. Determine la salida de cada uno de los siguientes circuitos, suponiendo que la entrada superior es 1 y que la entrada inferior es 0.

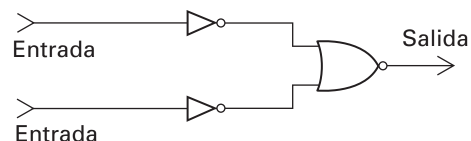

a.


b.

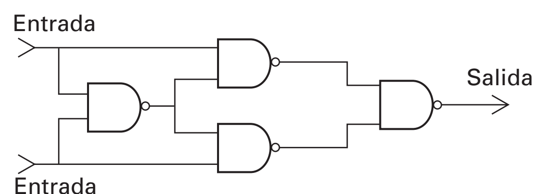

c.

¿Cuál sería la salida si la entrada superior fuera 0 y la entrada inferior 1?

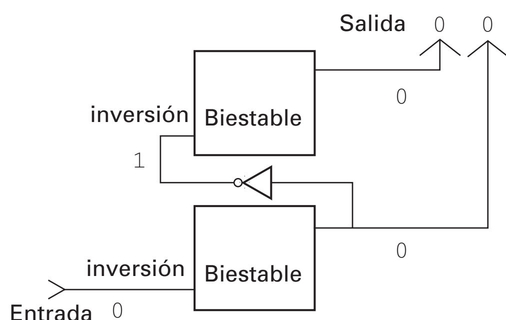
2. a. ¿Qué operación booleana calcula el siguiente circuito?



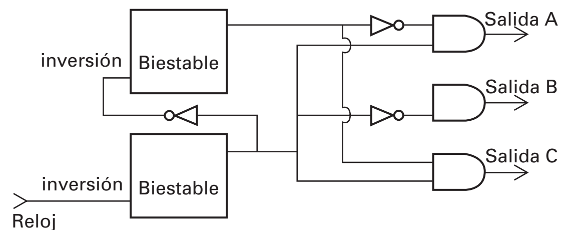
b. ¿Qué operación booleana calcula el siguiente circuito?



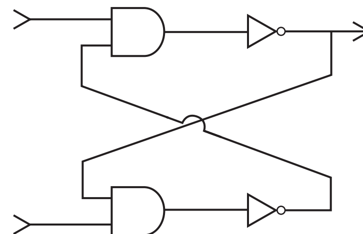
***3.** a. Si fuéramos a comprar un circuito biestable en una tienda de componentes electrónicos, nos encontraríamos con que el biestable puede tener una entrada adicional denominada *inversión*. Cuando esta entrada cambia de 0 a 1, la salida cambia de estado (si era un 0 pasa a ser 1 y viceversa). Sin embargo, cuando la entrada de inversión pasa de 1 a 0, no sucede nada. Aunque no conozcamos los detalles de la circuitería necesaria para conseguir este comportamiento, podríamos seguir utilizando este dispositivo como herramienta abstracta en otros circuitos. Considere el circuito que podríamos construir empleando dos de esos biestables. Si aplicamos un pulso a la entrada del circuito, el biestable inferior cambiará de estado. Sin embargo, el segundo biestable no cambiará, dado que su entrada (que se obtiene a partir de la salida de la puerta NOT) ha pasado de 1 a 0. Como resultado, ahora este circuito producirá las salidas 0 y 1. Un segundo pulso haría conmutar el estado de ambos biestables generando una salida de 1 y 0. ¿Cuál sería la salida después del tercer pulso? ¿Y después de un cuarto pulso?



b. A menudo es necesario coordinar las actividades de varios componentes de una computadora. Esto se consigue conectando una señal pulsante (denominada *señal de reloj*) a circuitos similares a los mostrados en el apartado a. Una serie de puertas adicionales (como se muestra en la figura siguiente) podrán entonces enviar señales de forma coordinada a otros circuitos conectados. Al estudiar este circuito, podrá confirmar que en los pulsos 1º, 5º, 9º... del reloj, se envía un 1 a la salida A. ¿Con qué pulsos del reloj se enviará un 1 a la salida B? ¿Con qué pulsos del reloj se enviará un 1 a la salida C? ¿A través de qué salida se enviará un 1 en el 4º pulso del reloj?



4. Suponga que las dos entradas del siguiente circuito están a 1. Describa lo que sucedería si la entrada superior cambiara temporalmente a 0. Describa lo que sucedería si se cambiara temporalmente la entrada inferior a 0. Vuelva a dibujar el circuito utilizando puertas NAND.



5. La siguiente tabla representa las direcciones y contenidos (en notación hexadecimal) de algunas celdas de la memoria principal de una máquina. Comenzando con esta disposición de memoria, siga la secuencia de instrucciones y escriba el contenido final de cada una de estas celdas de memoria.

Dirección	Contenido
30	FB
31	5D
32	AC
33	DC

Paso 1. Mueva el contenido de la celda cuya dirección es 32 a la celda cuya dirección es 33.

Paso 2. Mueva el valor 30 a la celda situada en la dirección 32.

Paso 3. Mueva el valor almacenado en la dirección 33 a la celda situada en la dirección 31.

6. ¿Por qué es necesaria la comprensión de datos? ¿Cuáles son los métodos de compresión de datos?

7. ¿Cuál es el valor del bit más significativo en los patrones de bits representados por los siguientes valores expresados en notación hexadecimal?

a. 5C   b. AF   c. 7C
d. 34   e. DF

8. ¿Cuál es el valor del bit menos significativo en los patrones de bits representados por los siguientes valores expresados en notación hexadecimal?

a. 7E   b. FD
c. 52   d. 3A

9. Exprese los siguientes patrones de bits en notación hexadecimal:

a. `101000001010`
b. `110001111011`
c. `000010111110`

10. Suponga que una cámara digital tiene una capacidad de almacenamiento de 512MB. ¿Cuántas fotografías podrían almacenarse en la cámara si cada una estuviera compuesta por 1024 píxeles por fila y 1024 píxeles por columna y cada píxel requiriera cuatro bytes de almacenamiento?

11. Suponga que una imagen está representada en una pantalla de una computadora mediante una matriz rectangular compuesta por 512 columnas y 2048 filas de píxeles. Si hacen falta 8 bits por píxel para codificar el color y otros 8 bits para codificar la intensidad, ¿cuántas celdas de memoria de tamaño igual a un byte hacen falta para almacenar la imagen completa?

12. a. Indique una ventaja de una unidad flash respecto al almacenamiento en la memoria principal.
b. Indique una ventaja que el almacenamiento en la memoria principal tiene con respecto a una unidad flash.

13. Suponga que solo están vacíos 340GB de los 500GB de disco duro que tiene su computadora personal. ¿Sería razonable utilizar una unidad flash de 4GB para almacenar una copia de seguridad de toda la información contenida en el disco? ¿Sería razonable emplear discos DVD? ¿Y una unidad flash de 8GB?

14. Si cada sector de un disco magnético contiene 512 bytes, ¿cuántos sectores son necesarios para almacenar una única página de texto (de 80 líneas de 120 caracteres) si representamos cada carácter en Unicode?

15. ¿Cuántos bytes de espacio de almacenamiento harían falta para almacenar una novela de 650 páginas en la que cada página contenga 3250 caracteres si se utiliza código ASCII? ¿Cuántos bytes harían falta si usáramos en su lugar código Unicode?

16. ¿Cuál es el tiempo típico de latencia de una unidad de disco duro que está girando a 360 revoluciones por segundo?

17. ¿Cuál es el tiempo medio de acceso para una unidad de disco duro que gire a 360 revoluciones por segundo, si el tiempo de búsqueda es de 10 milisegundos?

18. Suponga que un mecanógrafo puede escribir 100 palabras por minuto de manera continua, trabajando 24 horas al día y siete días a la semana. ¿Cuánto tardaría el mecanógrafo en llenar un DVD con una capacidad de 4,7GB? Suponga que cada palabra tiene siete caracteres y que cada carácter requiere un byte de almacenamiento.

19. He aquí un mensaje en ASCII. ¿Qué es lo que dice?

```
01001010 01110100 00100000 01101001
01110011 00100000 01110101 01101110
01101011 01101111 01100001 01110111
01101110 00100001
```

20. El mensaje siguiente está codificado en ASCII utilizando un byte por carácter. Luego se ha representado el mensaje en notación hexadecimal. ¿Qué es lo que dice el mensaje?

```
6E6F7469E6173636969
```

21. Codifique las siguientes frases en ASCII utilizando un byte por carácter.

a. Hello!
b. 20 / 5 = 4.

22. Exprese las siguientes frases en ASCII utilizando notación hexadecimal.
a. Hi!
b. 10 / 5 = 2.

23. Indique las representaciones binarias de los enteros comprendidos entre 12 y 22.

24. a. Escriba el número 45 representando el 4 y el 5 en ASCII.
b. Escriba el número 45 en representación binaria.

25. ¿Cúal es la utilidada de la distancia de Hamming en los códigos de corrección de errores?

***26.** Convierta cada una de las siguientes representaciones binarias a su representación equivalente en base diez:
a. 1111 b. 0001 c. 10101
d. 1000 e. 10011 f. 000000
g. 1001 h. 10001 i. 100001
j. 11001 k. 11010 l. 11011

***27.** Convierta cada una de las siguientes representaciones en base diez a su representación binaria equivalente:
a. 7 b. 11 c. 16
d. 17 e. 31

***28.** Convierta cada una de las siguientes representaciones exceso 16 a su representación en base diez equivalente:
a. 10001 b. 10101 c. 01101
d. 01111 e. 11111

***29.** Convierta cada una de las siguientes representaciones en base diez a su equivalente en exceso cuatro:
a. 0 b. 3 c. −2
d. −1 e. 2

***30.** Convierta cada una de las siguientes representaciones en complemento a dos a su representación equivalente en base diez:
a. 01111 b. 10100 c. 01100
d. 10000 e. 10110

***31.** Convierta cada una de las siguientes representaciones en base diez a su representación equivalente en complemento a dos, suponiendo que cada valor está representado mediante 7 bits:
a. 13 b. −13 c. −1
d. 0 e. 16

***32.** Realice cada una de las siguientes sumas, suponiendo que las cadenas de bits representan valores expresados en complemento a dos. Identifique todos los casos en los que la respuesta sea incorrecta a causa del desbordamiento.
a. 00101 +01000
b. 11111 +00001
c. 01111 +00001
d. 10111 +11010
e. 11111 +11111
f. 00111 +01100

***33.** Resuelva cada uno de los siguientes problemas traduciendo los valores en notación de complemento a dos (usando patrones de 5 bits), convirtiendo cada problema de resta en un problema de suma equivalente y efectuando dicha suma. Compruebe los resultados convirtiendo la respuesta a notación en base diez. (Tenga cuidado con los desbordamientos.)
a. 5 +1
b. 5 −1
c. 12 −5
d. 8 −7
e. 12 +5
f. 5 −11

***34.** Convierta cada una de las siguientes representaciones binarias a su representación equivalente en base diez:
a. 11.11 b. 100.0101 c. 0.1101
d. 1.0 e. 10.01

***35.** Exprese cada uno de los siguientes valores en notación binaria:
a. $5\frac{3}{4}$ b. $15\frac{15}{16}$ c. $5\frac{3}{8}$
d. $1\frac{1}{4}$ e. $6\frac{5}{8}$

***36.** Decodifique los siguientes patrones de bits utilizando el formato en punto flotante descrito en la Figura 1.26:
a. 01011001 b. 11001000
c. 10101100 d. 00111001

***37.** Codifique los siguientes valores utilizando el formato en punto flotante de 8 bits des-

crito en al Figura 1.26. Indique los casos en los que se produce un error de truncamiento.

a. $-7\frac{1}{2}$ b. $\frac{1}{2}$ c. $-3\frac{3}{4}$
d. $\frac{7}{32}$ e. $\frac{31}{32}$

***38.** Suponiendo que no estamos obligados a utilizar la forma normalizada, enumere todos los patrones de bits que podrían emplearse para representar el valor $\frac{3}{8}$ utilizando el formato de punto flotante descrito en la Figura 1.26.

***39.** ¿Cuál es la mejor aproximación a la raíz cuadrada de 2 que puede expresarse en el formato en punto flotante de 8 bits descrito en la Figura 1.26? ¿Qué valor se obtendría en realidad si halláramos el cuadrado de esta aproximación utilizando una máquina que emplee dicho formato de punto flotante?

***40.** ¿Cuál es la mejor aproximación del valor $\frac{1}{10}$ que puede representarse utilizando el formato en punto flotante de 8 bits descrito en la Figura 1.26?

***41.** Explique cómo pueden producirse errores al guardar en notación de punto flotante una serie de medidas realizadas utilizando el sistema métrico. Por ejemplo, ¿qué pasaría si registráramos el valor 110 cm en unidades de metro?

***42.** Uno de los patrones de bits 01011 y 11011 representa un valor almacenado en notación exceso 16, mientras que el otro representa el mismo valor almacenado en notación de complemento a dos.

a. ¿Qué es lo que podemos determinar acerca de este valor común?
b. ¿Cuál es la relación entre un patrón que representa un valor almacenado en complemento a dos y el patrón que representa el mismo valor almacenado en notación en exceso, cuando ambos sistemas utilizan la misma longitud de patrones de bits?

***43.** Los tres patrones de bits 10000010, 01101000 y 00000010 son representaciones del mismo valor en complemento a dos, notación en exceso y en el formato punto flotante de 8 bits mostrado en la Figura 1.26, pero no necesariamente en dicho orden, ¿cuál es el patrón común y qué patrón está en cada notación?

***44.** ¿Cuáles de los siguientes valores no pueden representarse de manera precisa en el formato de punto flotante presentado en la Figura 1.26?

a. $6\frac{1}{2}$ b. $\frac{13}{16}$ c. 9
d. $\frac{17}{32}$ e. $\frac{15}{16}$

***45.** Si cambiáramos la longitud de las cadenas de bits utilizadas para representar enteros en binario, de 4 bits a 6 bits, ¿qué variación se produciría en el valor del entero más grande que puede representarse? ¿Y en el caso de que estuviéramos utilizando notación en complemento a dos?

***46.** ¿Cuál sería la representación hexadecimal de la dirección de memoria más alta en una memoria de 4MB si cada celda tiene una capacidad de un byte?

***47.** ¿Cuál sería la versión codificada del mensaje

`xxy yyx xxy xxy yyx`

si se utilizara compresión LZW, comenzando con el diccionario que contiene los símbolos *x*, *y* y espacio (como se describe en la Sección 1.8)?

***48.** El siguiente mensaje se ha comprimido utilizando compresión LZW y un diccionario cuya primera, segunda y tercera entradas son *x*, *y* y el carácter de espacio, respectivamente. ¿Cuál es el mensaje descomprimido?

`22123113431213536`

***49.** Si se comprime el mensaje

`xxy yyx xxy xxyy`

utilizando LZW con un diccionario de partida cuya primera, segunda y tercera entradas son *x*, *y* y espacio, respectivamente, ¿cuáles serían las entradas en el diccionario final?

***50.** Como veremos en el siguiente capítulo, un método de transmitir bits a través de siste-

mas telefónicos tradicionales consiste en convertir los patrones de bits a sonido, transferir el sonido a través de las líneas telefónicas y luego convertir de nuevo el sonido en patrones de bits. Estas técnicas están limitadas a tasas de transferencia de 57,6 Kbps. ¿Es esto suficiente para mantener una teleconferencia si comprimimos el vídeo utilizando MPEG?

***51.** Codifique las siguientes frases en ASCII utilizando paridad par, añadiendo un bit de paridad en el extremo de mayor peso de cada código de carácter:
a. Does 100/5 = 20?
b. The total cost is $7.25.

***52.** El siguiente mensaje se transmitió originalmente con paridad impar para cada una de las cadenas de bits. ¿En qué cadenas podemos deducir que se han producido errores?

`11001 11011 10110 00000 11111`
`10001 10101 00100 01110`

***53.** Suponga que generamos un código de 24 bits representando cada símbolo mediante tres copias consecutivas de su representación ASCII (por ejemplo, el símbolo A se representaría mediante la cadena de bits 010000010100000101000001). ¿Qué propiedades de corrección de errores tiene este nuevo código?

***54.** Utilizando el código de corrección de errores descrito en la Figura 1.30, decodifique las siguientes palabras:
a. `111010 110110`
b. `101000 100110 001100`
c. `011101 000110 000000 010100`
d. `010010 001000 001110 101111 000000 110111 100110`
e. `010011 000000 101001 100110`

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Se ha producido un error de truncamiento en una situación crítica, provocando grandes daños y pérdida de vidas. ¿Quién es el responsable, si es que hay alguien? ¿El diseñador del hardware? ¿El diseñador del software? ¿El programador que escribió esa parte del programa? ¿La persona que decidió utilizar el software en esa aplicación concreta? ¿Qué pasa si el software hubiera sido corregido por la empresa que lo desarrolló originalmente, pero esa actualización no hubiera sido adquirida y aplicada a esa aplicación crítica concreta?¿Y qué sucede si el software hubiera sido pirateado?
2. ¿Es aceptable que una persona ignore la posibilidad de que existan errores de truncamiento y sus consecuencias a la hora de desarrollar sus propias aplicaciones?
3. ¿Fue ético desarrollar software en la década de 1970 utilizando solo dos dígitos para representar el año (como por ejemplo utilizar 76 para representar el año 1976), ignorando el hecho de que el software presentaría fallos en cuanto se llegara al fin de siglo. ¿Resulta ético emplear hoy día únicamente tres dígitos para representar el año (como por ejemplo, usar 982 para 1982 y 015 para 2015)? ¿Y en el caso de usar solo cuatro dígitos?

4. Muchas personas sostienen que al codificar la información se diluye o distorsiona dicha información, ya que se obliga en primer lugar a cuantificarla. Esas personas argumentan que todos los cuestionarios en los que se pide al entrevistado que escriba su opinión respondiendo mediante una escala de uno a cinco están viciados desde el principio. ¿Hasta qué punto es cuantificable la información? ¿Pueden cuantificarse los pros y los contras de las diferentes ubicaciones posibles para la instalación de una planta de residuos? ¿Es cuantificable el debate acerca de la energía nuclear y de los residuos nucleares? ¿Es peligroso basar las decisiones en promedios y otros resultados estadísticos? ¿Es ético que las agencias de noticias informen acerca de los resultados de las encuestas sin incluir la formulación exacta de las preguntas planteadas? ¿Es posible cuantificar el valor de una vida humana? ¿Resulta aceptable que una empresa deje de invertir en la mejora de un producto, aún cuando esa inversión adicional pudiera reducir la probabilidad de que se produzcan muertes relacionadas con el uso de ese producto?
5. ¿Debería haber una distinción en lo relativo a recopilar y difundir datos dependiendo de la forma de esos datos? En otras palabras, ¿los derechos para recopilar y difundir fotografías, audio o vídeo deberían ser iguales que los derechos de recopilar y difundir textos?
6. Independientemente de que esa su intención o no, toda noticia escrita por un periodista suele reflejar la visión subjetiva de ese periodista. A menudo, con solo cambiar unas cuantas palabras, se puede dar a una historia una connotación positiva o negativa. (Compare, por ejemplo, los siguientes dos titulares: “La mayoría de los encuestados se opone al referendum” y “Una parte significativa de las personas encuestadas apoya el referendum”.) ¿Existe alguna diferencia entre modificar una noticia (omitiendo ciertos datos o seleccionando cuidadosamente las palabras) y modificar una fotografía?
7. Suponga que el uso de un sistema de compresión de datos provoca la pérdida de una serie de sutiles pero significativos elementos de información. ¿Qué problemas de responsabilidad civil podrían surgir? ¿Cómo habría que resolverlos?

Lecturas adicionales

Drew, M. y Z. Li. *Fundamentals of Multimedia.* Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsall, F. *Multimedia Communications.* Boston, MA: Addison-Wesley, 2001.

Hamacher, V. C., Z. G. Vranesic y S. G. Zaky. *Computer Organization,* 5ª ed. Nueva York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming,* Vol. 2, 3ª ed. Boston, MA: Addison-Wesley, 1998.

Long, B. *Complete Digital Photography,* 3ª ed. Hingham, MA: Charles River Media, 2005.

Miano, J. *Compressed Image File Formats*. Nueva York: ACM Press, 1999.

Petzold, C. *CODE: The Hidden Language of Computer Hardware and Software*. Redman, WA: Microsoft Press, 2000.

Salomon, D. *Data Compression: The Complete Reference,* 4ª ed. Nueva York: Springer, 2007.

Sayood, K. *Introduction to Data Compression,* 3ª ed. San Francisco: Morgan Kaufmann, 2005.

CAPÍTULO

2

Tratamiento de datos

En este capítulo veremos cómo manipula los datos una computadora y cómo se comunica con dispositivos periféricos tales como impresoras y teclados. Además, exploraremos los fundamentos de la arquitectura de computadoras y veremos cómo se programan mediante instrucciones codificadas, denominadas instrucciones en lenguaje máquina.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

En el Capítulo 1 hemos estudiado temas relacionados con el almacenamiento de datos en una computadora. En este capítulo veremos cómo una computadora manipula dichos datos. Esta manipulación consiste en transferir los datos de una posición a otra, así como en la realización de operaciones como por ejemplo cálculos aritméticos, edición de textos y manipulación de imágenes. Comenzaremos analizando con más detalle la arquitectura de las computadoras, yendo más allá de los sistemas de almacenamiento de datos.

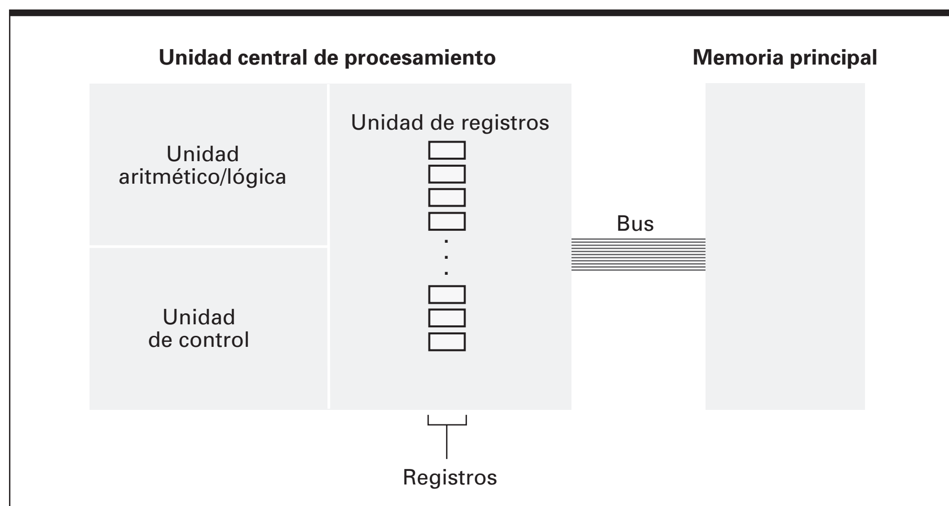
2.1 Arquitectura de computadoras

La circuitería de una computadora que controla el tratamiento de los datos se conoce como **unidad central de procesamiento (CPU, Central Processing Unit)**, y a menudo simplemente se denomina procesador. En las máquinas de mediados del siglo xx, las CPU eran de gran tamaño, compuestas en ocasiones por varios bastidores de circuitos electrónicos, lo que reflejaba la importancia de dicha unidad. Sin embargo, los avances tecnológicos han permitido reducir enormemente el tamaño de estos dispositivos. Los procesadores que podemos encontrar hoy día en las computadoras de sobremesa y portátiles se encapsulan en pequeños cuadrados planos (de aproximadamente 2,5 por 2,5 centímetros) cuyos pines de conexión encajan en un zócalo que está montado sobre la tarjeta de circuito principal de la máquina (la **tarjeta madre**). En los teléfonos inteligentes, miniportátiles y otros dispositivos de Internet móviles (**MID**, *Mobile Internet Devices*), los procesadores suelen tener la mitad del tamaño de un sello de correos. Debido a su pequeño tamaño, estos procesadores se denominan **microprocesadores**.

El procesador

Un procesador consta de tres partes (Figura 2.1): la **unidad aritmético/lógica**, que contiene los circuitos que realizan las operaciones con los datos (como por

Figura 2.1 El procesador y la memoria principal conectados a través de un bus.



ejemplo sumas y restas), la **unidad de control**, que contiene los circuitos que coordinan las actividades de la máquina y la **unidad de registros**, que contiene celdas de almacenamiento de datos (similares a las celdas de la memoria principal), denominadas **registros**, que se emplean para almacenar temporalmente la información dentro del procesador.

Algunos de los registros de la unidad de registro se consideran **registros de uso general** mientras que otros son **registros de uso especial**. En la Sección 2.3 hablaremos de algunos de los registros de uso especial. Por el momento, vamos a centrarnos en los registros de uso general.

Los registros de uso general sirven como lugares de almacenamiento temporal para los datos que están siendo tratados por el procesador. Estos registros almacenan las entradas a la circuitería de la unidad aritmético/lógica y proporcionan espacio de almacenamiento para los resultados generados por dicha unidad. Para realizar una operación con datos almacenados en la memoria principal, la unidad de control transfiere los datos desde la memoria hasta los registros de uso general, informa a la unidad aritmético/lógica de qué registros son los que contienen los datos, activa los circuitos apropiados dentro de la unidad aritmético/lógica y le dice a esta en qué registro debe almacenar el resultado.

Para transferir los patrones de bits, el procesador y la memoria principal de una máquina se conectan a través de un conjunto de hilos de conexión, denominado **bus** (véase de nuevo la Figura 2.1). A través de este bus, el procesador extrae (lee) datos de la memoria principal, suministrando la dirección de la celda de memoria pertinente, junto con una señal electrónica que le indica a los circuitos de memoria que debe extraer los datos contenidos en la celda indicada. De forma similar, el procesador coloca (escribe) datos en la memoria proporcionando la dirección de la celda de destino y los datos que hay que almacenar, junto con la señal electrónica apropiada que le dice a la memoria principal que debe almacenar los datos que se le están enviando.

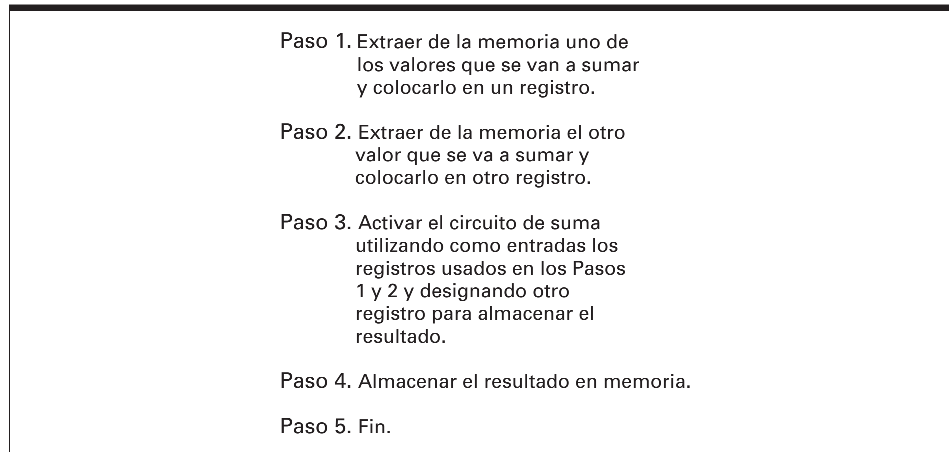
Basándonos en este diseño, la tarea de sumar dos valores almacenados en la memoria principal implica algunas tareas más que la mera ejecución de la operación de suma. Es necesario transferir los datos desde la memoria principal hasta los registros contenidos en el procesador; luego hay que sumar los valores y colocar el resultado en un registro y, finalmente, el resultado debe almacenarse en una celda de memoria. El proceso completo se resume en los cinco pasos indicados en la Figura 2.2.

El concepto de programa almacenado

Las primeras computadoras no destacaban precisamente por su flexibilidad: los pasos que cada dispositivo ejecutaba estaban integrados, como parte de la máquina, en la propia unidad de control. Para conseguir una mayor flexibilidad, algunas de las primeras computadoras electrónicas se diseñaron de forma que el procesador pudiera volverse a cablear fácilmente. Esta flexibilidad se conseguía por medio de unos zócalos de interconexión similares a los paneles de conmutación de las antiguas centrales telefónicas, en los que se introducían en una serie de agujeros los extremos de los cables de interconexión.

Uno de los mayores avances (atribuido, aparentemente de forma incorrecta, a John von Neumann) se produjo cuando los diseñadores de computadoras se dieron cuenta de que un programa también podía codificarse y

Figura 2.2 Suma de valores almacenados en la memoria.



almacenarse en la memoria principal como si fuera cualquier otro tipo de dato. Si se diseña la unidad de control para que extraiga el programa de la memoria, decodifique las instrucciones y las ejecute, puede modificarse el programa ejecutado por la máquina simplemente cambiando el contenido de la memoria de la computadora en lugar de tener que recablear el procesador.

La idea de almacenar el programa de una computadora en su memoria principal se conoce como **concepto de programa almacenado** y se ha convertido en el método estándar utilizado en la actualidad, tan estándar, que parece obvio. Lo que hizo difícil originalmente llegar a este concepto fue el hecho de que todo el mundo pensaba en los programas y en los datos como si fueran entidades distintas. Los datos se almacenaban en la memoria mientras

Memoria caché

Es instructivo comparar los distintos recursos de memoria de una computadora en relación con su funcionalidad. Los registros se utilizan para almacenar los datos que son inmediatamente aplicables a la operación que se esté llevando a cabo; la memoria principal se emplea para almacenar los datos que se necesitarán en el próximo futuro y el almacenamiento masivo se utiliza para almacenar datos que probablemente no vayan a ser necesarios en el futuro inmediato. Muchas máquinas están diseñadas con un nivel de memoria adicional, llamada **memoria caché**. La memoria caché es una parte (quizá de varios cientos de KB) de memoria de alta velocidad localizada dentro del propio procesador. En esta área de memoria especial, la máquina trata de mantener una copia de aquella porción de la memoria principal que es de interés en ese momento concreto. Con este tipo de diseño, las transferencias de datos que normalmente se llevarían a cabo entre los registros y la memoria principal se realizan entre los registros y la memoria caché. Todos los cambios realizados en la memoria caché se transfieren luego de manera colectiva a la memoria principal, en algún momento más oportuno. El resultado es un procesador que puede ejecutar su ciclo de máquina más rápidamente, porque no se ve retardado por la comunicación con la memoria principal.

que los programas eran parte del procesador. El resultado de esta forma de pensar constituye un ejemplo palmario de situación en la que los árboles no dejan ver el bosque. Es fácil verse atrapado en este tipo de prejuicios erróneos y puede que el desarrollo de las Ciencias de la computación tenga todavía muchos de estos prejuicios sin que ni siquiera seamos conscientes de ellos. De hecho, parte del interés que la ciencia despierta se debe a que los nuevos conceptos están constantemente abriendo puertas hacia nuevas teorías y aplicaciones.

Cuestiones y ejercicios

1. ¿Qué secuencia de sucesos cree que harían falta para desplazar el contenido de una celda de memoria de una computadora a otra celda de memoria?
2. ¿Qué información debe suministrar el procesador a los circuitos de la memoria principal para escribir un valor en una celda de memoria?
3. Tanto el almacenamiento masivo como la memoria principal y los registros de uso general son sistemas de almacenamiento. ¿Qué diferencia hay en el uso que se da a estos distintos sistemas?

2.2 Lenguaje máquina

Con el fin de aplicar el concepto de programa almacenado, los procesadores están diseñados para reconocer instrucciones codificadas como patrones de bits. Este conjunto de instrucciones junto con el sistema de codificación utilizado forman lo que se conoce como **lenguaje máquina**. Una instrucción expresada en este lenguaje se denomina **instrucción de nivel máquina**.

Repertorio de instrucciones

La lista de instrucciones de lenguaje máquina que un procesador típico es capaz de decodificar y ejecutar es bastante corta. De hecho, una vez que una máquina puede realizar ciertas tareas elementales pero convenientemente elegidas, el añadir más funcionalidad no incrementa las capacidades teóricas de la máquina. En otras palabras, más allá de un cierto punto, la funcionalidad adicional puede proporcionar algo más de comodidad, pero no añade nada a las capacidades fundamentales de la máquina.

El grado con el que el diseño de un máquina debe aprovechar este hecho ha conducido a dos filosofías distintas de arquitecturas de procesador. Una de ellas es que un procesador debe diseñarse para ejecutar un conjunto mínimo de instrucciones en lenguaje máquina. Esta técnica conduce a lo que se denomina arquitectura **RISC** (***Reduced Instruction Set Computer***, Computadora de conjunto reducido de instrucciones). El argumento en favor de la arquitectura RISC es que las máquinas de ese tiempo son eficientes, rápidas y más baratas

¿Quién inventó el qué?

Asignar a una sola persona todo el mérito de un invento suele ser bastante dudoso. A Thomas Edison se le reconoce la invención de la lámpara incandescente, pero otros investigadores estaban desarrollando lámparas similares y podríamos considerar, en un cierto sentido, que Edison tuvo suerte en ser el que obtuvo la patente. A los hermanos Wright se les reconoce la invención del aeroplano, pero competían con otros muchos personajes contemporáneos y se beneficiaron del trabajo realizado por ellos, todos los cuales les deben algo, a su vez, a Leonardo da Vinci, que jugueteó con la idea de máquinas voladoras ya en el siglo XV. Incluso los diseños de Leonardo estaban aparentemente basados en ideas anteriores. Por supuesto, en estos casos el inventor reconocido sigue pudiendo reclamar legítimamente el mérito que le corresponde. En otros casos, la historia parece haber concedido ese mérito de manera inapropiada, un ejemplo sería el concepto de programa almacenado. Sin ninguna duda, John von Neumann era un científico brillante que se merece un reconocimiento por sus numerosas contribuciones, pero una de las contribuciones por las que la historia popular ha elegido concederle el mérito, el concepto de programa almacenado, fue aparentemente desarrollado por un equipo de investigadores dirigido por J. P. Eckert en la Escuela Moore de Ingeniería Eléctrica de la universidad de Pensilvania. John von Neumann fue simplemente el primero en publicar un trabajo en el que informaba acerca de dicha idea, lo cual es la razón de que en el campo de la mitología informática se le haya seleccionado como el inventor de ese concepto.

de fabricar. Por el contrario, otros diseñadores argumentan en favor de procesadores que tengan la capacidad de ejecutar un gran número de instrucciones complejas, aún cuando muchas de ellas sean técnicamente redundantes. El resultado de este enfoque se conoce con el nombre de **CISC** (***Complex Instruction Set Computer***, Computadora de conjunto complejo de instrucciones). El argumento en favor de la arquitectura CISC es que los procesadores más complejos pueden enfrentarse mejor a la complejidad cada vez mayor del software actual. Con una arquitectura CISC, los programas pueden aprovecharse de la existencia de un conjunto rico y potente de instrucciones, muchas de las cuales requerirían una secuencia multi-instrucción en un diseño RISC.

En la década de 1990 y en la primera década de este milenio, los procesadores CISC y RISC comercialmente disponibles han estado compitiendo activamente por el papel predominante en el campo de los equipos de sobremesa. Los procesadores de Intel, utilizados en los PC, son un ejemplo de arquitectura CISC; los procesadores PowerPC (desarrollados mediante una alianza entre Apple, IBM y Motorola) son ejemplos de arquitectura RISC y fueron utilizados en el Apple Macintosh. A medida que ha ido pasando el tiempo, el coste de fabricación de los procesadores CISC se ha reducido enormemente; por ello, los procesadores de Intel (o sus equivalentes de AMD, Advanced Micro Devices, Inc.) pueden ahora encontrarse en todas las computadoras de sobremesa y portátiles (incluso Apple está ahora construyendo computadoras basadas en los productos de Intel).

Aunque la arquitectura CISC se ha garantizado un puesto predominante en las computadoras de sobremesa, tiene un insaciable apetito de potencia eléc-

trica. Por el contrario, la empresa Advanced RISC Machine (ARM) ha diseñado una arquitectura RISC específicamente pensada para un bajo consumo. (Advanced RISC Machine fue originalmente Acorn Computers y ahora se conoce como ARM Holdings.) Por tanto, los procesadores basados en el diseño de ARM y que son fabricados por diversas empresas, entre las que se incluyen Qualcomm y Texas Instruments, están presentes hoy día en controladoras de juegos, televisiones digitales, sistemas de navegación, módulos para automoción, teléfonos celulares, teléfonos inteligentes y otros dispositivos de electrónica de consumo.

Independientemente de la elección que se haga entre RISC y CISC, las instrucciones de una máquina pueden clasificarse en tres grupos: (1) el grupo de transferencia de datos, (2) el grupo aritmético/lógico y (3) el grupo de control.

Transferencia de datos El grupo de transferencia de datos está compuesto por instrucciones que solicitan el movimiento de datos desde una ubicación a otra. Los pasos 1, 2 y 4 de la Figura 2.2 caen dentro de esta categoría. Es preciso recalcar que el uso de términos tales como ***transferir*** o ***mover*** para identificar a este grupo de instrucciones es en realidad engañoso. Es raro que los datos que se están transfiriendo se borren de su ubicación original. El proceso implicado en una instrucción de transferencia es más una copia de los datos que un movimiento de los mismos. Por tanto, otros términos como ***copiar*** o ***clonar*** permitirían describir mejor las acciones llevadas a cabo por este grupo de instrucciones.

Sin salirnos del campo de la terminología, debemos mencionar que suelen emplearse términos especiales a la hora de hacer referencia a la transferencia de datos entre el procesador y la memoria principal. Una solicitud para llenar un registro de uso general con el contenido de una celda de memoria se suele denominar instrucción LOAD (instrucción de carga); a la inversa, una solicitud para transferir el contenido de un registro a una celda de memoria se denomina instrucción STORE (instrucción de almacenamiento). En la Figura 2.2, los pasos 1 y 2 especifican instrucciones LOAD y el Paso 4 indica una instrucción STORE.

Instrucciones de longitud variable

Para simplificar las explicaciones a lo largo del texto, el lenguaje máquina utilizado para los ejemplos de este capítulo (y descrito en el Apéndice C) utiliza un tamaño fijo (dos bytes) para todas las instrucciones. Por tanto, para cargar una instrucción, el procesador siempre extrae el contenido de dos celdas de memoria consecutivas e incrementa el contador de programa en dos unidades. Este comportamiento predecible simplifica la tarea de carga de las instrucciones y es característica de las máquinas RISC. Sin embargo, las máquinas CISC tienen lenguajes máquina con instrucciones de longitud variable. Los procesadores de Intel actuales, por ejemplo, tienen instrucciones que van desde las de un único byte hasta otras de múltiples bytes cuya longitud depende de la utilización exacta de dicha instrucción. Los procesadores con este tipo de lenguaje máquina determinan la longitud de instrucción que hay que cargar analizando el código de operación de dicha instrucción. Es decir, el procesador carga primero el código de operación de la instrucción y luego, dependiendo del patrón de bits recibido, sabe cuántos más bits debe cargar de la memoria para obtener el resto de la instrucción.

Un grupo importante de instrucciones dentro de la categoría de transferencia de datos está formado por los comandos utilizados para comunicarse con dispositivos externos al contexto definido por el procesador y la memoria principal (impresoras, teclados, pantallas, unidades de disco, etc.). Puesto que estas instrucciones se encargan de generar las actividades de entrada/salida, E/S (I/O, Input/Output), de la máquina se denominan **instrucciones de E/S** (o, en inglés, instrucciones I/O) y, en ocasiones, se las considera una categoría de instrucciones diferente. Por otro lado, en la Sección 2.5 se describe cómo pueden gestionarse estas actividades de E/S mediante el mismo conjunto de instrucciones que solicita la transferencia de datos entre el procesador y la memoria principal. Por tanto, vamos a considerar las instrucciones de E/S como parte del grupo de transferencia de datos.

Aritmético/Lógico El grupo aritmético/lógico está compuesto por aquellas instrucciones que le dicen a la unidad de control que debe solicitar una cierta actividad dentro de la unidad aritmético/lógica. El paso 3 de la Figura 2.2 cae dentro de este grupo. Como su propio nombre sugiere, la unidad aritmético/lógica puede realizar también otras operaciones diferentes de las operaciones aritméticas básicas. Algunas de estas operaciones adicionales básicas son las operaciones booleanas AND, OR y XOR, presentadas en el Capítulo 1 y que analizaremos con más detalle posteriormente.

Otro conjunto de operaciones disponible dentro de la mayor parte de las unidades aritmético/lógicas permite pasar el contenido de los registros hacia la derecha o hacia la izquierda sin salir del propio registro. Estas operaciones se conocen con el nombre de operaciones SHIFT (desplazamiento) o ROTATE (rotación), dependiendo de si los bits que se "caen" por el extremo del registro se descartan simplemente (SHIFT) o se utilizan para rellenar el hueco que queda en el otro extremo (ROTATE).

Control El grupo de control está compuesto por aquellas instrucciones que dirigen la ejecución del programa en lugar de la manipulación de los datos. El paso 5 de la Figura 2.2 cae dentro de esta categoría, aunque se trata de un ejemplo

Figura 2.3 División de valores almacenados en la memoria.

Paso 1. CARGAR (LOAD) un registro con un valor de la memoria.

Paso 2. CARGAR (LOAD) otro registro con otro valor de la memoria.

Paso 3. Si este segundo valor es cero, SALTAR (JUMP) al Paso 6.

Paso 4. Dividir el contenido del primer registro entre el segundo registro y colocar el resultado en un tercer registro.

Paso 5. ALMACENAR (STORE) el contenido del tercer registro en memoria.

Paso 6. STOP.

bastante elemental. Este grupo contiene muchas de las instrucciones más interesantes del repertorio de una máquina, como la familia de instrucciones JUMP (o BRANCH), instrucciones de salto o bifurcación, que se utilizan para ordenar al procesador que ejecute una instrucción distinta de la que se encuentra a continuación en la lista. Existen dos tipos de instrucciones JUMP: **saltos incondicionales** y **saltos condicionales**. Un ejemplo de salto incondicional sería la instrucción "Saltar al paso 5"; mientras que un ejemplo de salto condicional sería: "Si el valor obtenido es 0, entonces saltar al paso 5." La diferencia es que un salto condicional solo provoca el salto si se satisface una cierta condición. Por ejemplo, la secuencia de instrucciones de la Figura 2.3 representa un algoritmo para dividir dos valores, en el que el paso 3 es un salto condicional que nos protege frente a la posibilidad de división por cero.

Un ejemplo de lenguaje máquina

Vamos a ver ahora cómo se codifican las instrucciones de una computadora típica. La máquina que vamos a utilizar para nuestro análisis se describe en el Apéndice C y se ilustra en la Figura 2.4. Dispone de 16 registros de uso general y de 256 celdas en la memoria principal, cada una de ellas con una capacidad de 8 bits. Para propósitos de referencia, vamos a etiquetar los registros con los valores de 0 a 15 y las direcciones de las celdas de memoria con los valores 0 a 255. Por comodidad, vamos a considerar que estas etiquetas y direcciones son valores representados en base dos y vamos a expresar los patrones de bits resultantes en notación hexadecimal. Etiquetaremos los registros de 0 a F y las direcciones de las celdas de memoria de 00 a FF.

La versión codificada de una instrucción en lenguaje máquina está compuesta de dos partes: el campo de **código de operación** y el campo **operando**. El patrón de bits que aparece en el campo correspondiente al código de operación nos indica cuál es la operación elemental (como por ejemplo STORE, SHIFT, XOR o JUMP) solicitada por la instrucción. Los patrones de bits contenidos en el campo operando proporcionan información más detallada acerca de la opera-

Figura 2.4 Arquitectura de la máquina descrita en el Apéndice C.

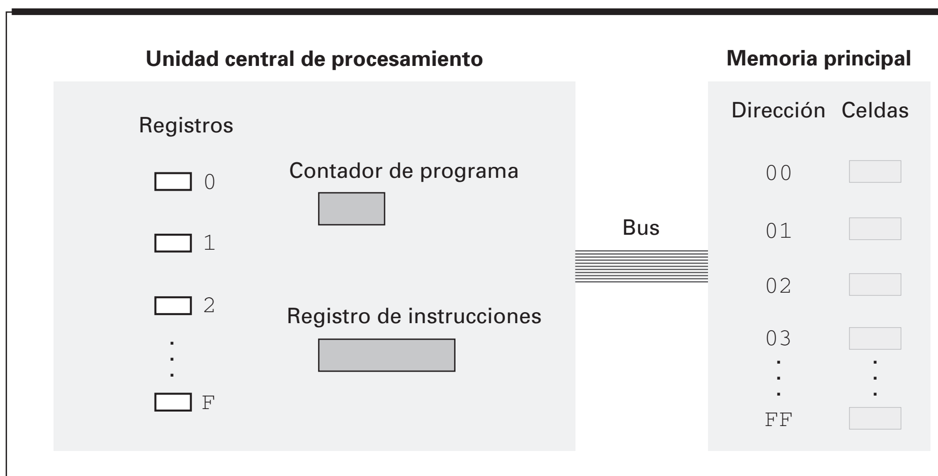


Figura 2.5 Composición de una instrucción para la máquina del Apéndice C.

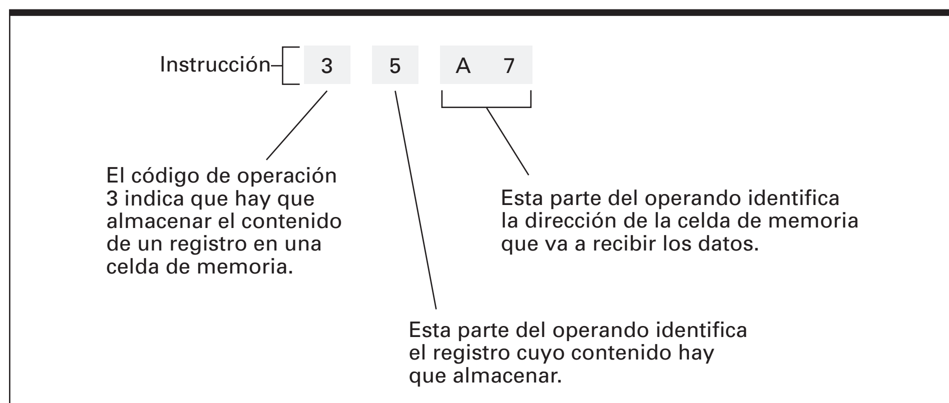


ción especificada por el código de operación. Por ejemplo, en el caso de una operación STORE, la información del campo operando indica qué registro contiene el dato que hay que almacenar y qué celda de memoria tiene que recibir el dato.

El lenguaje máquina completo de nuestra máquina de ejemplo (Apéndice C) está compuesto por solo dos instrucciones básicas. Cada una de estas instrucciones se codifica utilizando un total de 16 bits, representados mediante cuatro dígitos hexadecimales (Figura 2.5). El código de operación de cada instrucción está compuesto por los primeros cuatro bits (o lo que es lo mismo, el primer dígito hexadecimal. Observe (Apéndice C) que estos códigos de operación están representados por los dígitos hexadecimales 1 a C. En particular, la tabla del Apéndice C muestra que una instrucción que comienza con el dígito hexadecimal 3 hace referencia a la instrucción STORE, mientras que una instrucción que comienza con el dígito hexadecimal A hace referencia a la instrucción ROTATE.

El campo de operando de cada instrucción de nuestra máquina de ejemplo está compuesto por tres dígitos hexadecimales (12 bits) y, en todos los casos, excepto para la instrucción HALT (que es la instrucción de detención y no necesita ningún detalle adicional), permite clarificar la instrucción general especificada por el código de operación. Por ejemplo (Figura 2.6), si el primer dígito hexadecimal de una instrucción fuera 3 (el código de operación para almacenar el contenido de un registro), el siguiente dígito hexadecimal de la instrucción nos indicaría qué registro es el que hay que almacenar y los dos

Figura 2.6 Decodificación de la instrucción 35A7.



últimos dígitos hexadecimales nos dirían en qué celda de memoria hay que almacenar ese dato. Por tanto, la instrucción 35A7 (hexadecimal) se traduce en la instrucción "Almacenar (STORE) el patrón de bits contenido en el registro 5, depositándolo en la celda de memoria cuya dirección es A7". (Observe cómo simplifica las explicaciones el uso de la notación hexadecimal. En realidad, la instrucción 35A7 es el patrón de bits 0011010110100111.)

La instrucción 35A7 también proporciona un ejemplo explícito de por qué la capacidad de la memoria principal se mide en potencias de dos. Puesto que hemos reservado 8 bits de la instrucción para especificar la celda de memoria utilizada por la instrucción, es posible referenciar exactamente 2^8 celdas diferentes. Esto nos fuerza a construir una memoria principal que tenga exactamente este número de celdas, cuyas direcciones irán de 0 a 255. Si la memoria principal tuviera más celdas, no podríamos escribir instrucciones que distinguieran unas celdas de otras. Por el contrario, si la memoria principal tuviera menos celdas, existiría la posibilidad de escribir instrucciones que hicieran referencia a celdas inexistentes.

Veamos otro ejemplo de cómo se emplea el campo operando para clarificar la instrucción general dada por el código de operación: considere una instrucción con el código de operación 7 (hexadecimal), que solicita llevar a cabo la operación OR con el contenido de dos registros (ya veremos lo que significa combinar mediante OR dos registros en la Sección 2.4. Por ahora, lo que nos interesa simplemente es el modo en que las instrucciones se codifican). En este caso, el siguiente dígito hexadecimal indica el registro en el que hay que almacenar el resultado, mientras que los dos últimos dígitos hexadecimales indican cuáles son los dos registros que hay que combinar mediante OR. Por tanto, la instrucción 70C5 se traduce en la instrucción "combinar mediante OR el contenido del registro C y el contenido del registro 5 y almacenar el resultado en el registro 0".

Existe una distinción sutil entre las dos instrucciones LOAD de nuestra máquina. Aquí podemos ver que el código de operación 1 (hexadecimal) identifica una instrucción que carga un registro con el contenido de una celda de memoria, mientras que el código de operación 2 (hexadecimal) identifica una instrucción que carga un registro con un cierto valor concreto. La diferencia es que el campo de operación en una instrucción del primer tipo contendrá una dirección, mientras que en el segundo tipo, el campo de operando contendrá el propio patrón de bits que hay que cargar.

Observe que la máquina dispone de dos instrucciones ADD: una para sumar representaciones en complemento a dos y otra para sumar representaciones en punto flotante. Esta distinción es una consecuencia del hecho de que la suma de patrones de bits que representan valores codificados en notación en complemento a dos requiere llevar a cabo, dentro de la unidad aritmético/lógica, una serie de actividades diferente de las que son necesarias para sumar valores en notación de punto flotante.

Vamos a cerrar esta sección examinando la Figura 2.7, que contiene una versión codificada de las instrucciones de la Figura 2.2. Hemos supuesto que los valores que hay sumar están almacenados en notación de complemento a dos en las direcciones de memoria 6C y 6D y que la suma hay que almacenarla en la celda de memoria situada en la dirección 6E.

Figura 2.7 Versión codificada de las instrucciones de la Figura 2.2.

Instrucciones codificadas	Traducción
156C	Cargar el registro 5 con el patrón de bits almacenado en la celda de memoria situada en la dirección 6C.
166D	Cargar el registro 6 con el patrón de bits almacenado en la celda de memoria situada en la dirección 6D.
5056	Sumar el contenido del registro 5 con el del registro 6 como si fueran representaciones en complemento a dos y almacenar el resultado en el registro 0.
306E	Almacenar el contenido del registro 0 en la celda de memoria situada en la dirección 6E.
C000	Parar.

Cuestiones y ejercicios

1. ¿Por qué el término *movimiento* podría considerarse incorrecto para la operación de mover datos de una ubicación a otra, dentro de la máquina?
2. En el texto, las instrucciones JUMP se expresaban identificando explícitamente el destino, indicando el nombre (o número de paso) de dicho destino dentro de la instrucción JUMP (por ejemplo, "Saltar al Paso 6"). Una desventaja de esta técnica es que si el nombre (o el número) se cambia posteriormente, nos vemos obligados a localizar todos los saltos a dicha instrucción y cambiar también el nombre. Describa otra forma de expresar una instrucción JUMP de modo que no haya que indicar explícitamente el nombre del destino.
3. ¿A qué categoría pertenece la instrucción "Si 0 es igual a 0, entonces saltar al Paso 7", a la de los saltos condicionales o incondicionales? Explique su respuesta.
4. Escriba el programa de ejemplo de la Figura 2.7 utilizando patrones de bits reales.
5. Observe las siguientes instrucciones escritas en el lenguaje máquina descrito en el Apéndice C. Reescriba esas instrucciones en español normal.

   a. 368A **b.** BADE **c.** 803C **d.** 40F4
6. ¿Cuál es la diferencia entre las instrucciones 15AB y 25AB en el lenguaje máquina del Apéndice C?

7. He aquí algunas instrucciones expresadas en español normal. Traduzca cada una de ellas al lenguaje máquina descrito en el Apéndice C.

a. Cargar (LOAD) el registro número 3 con el valor hexadecimal 56.
b. Rotar (ROTATE) el registro número 5 tres bits hacia la derecha.
c. Combinar mediante AND el contenido del registro A y el contenido del registro 5 y almacenar el resultado en el registro 0.

2.3 Ejecución de programas

Las computadoras ejecutan un programa almacenado en su memoria copiando las instrucciones desde la memoria al procesador según va siendo necesario. Una vez que están en el procesador, cada instrucción se decodifica y se hace lo que la instrucción ordene. El orden en el que las instrucciones se extraen de la memoria se corresponde con el orden en el que están almacenadas en la memoria, a no ser que ese orden se altere mediante una instrucción JUMP.

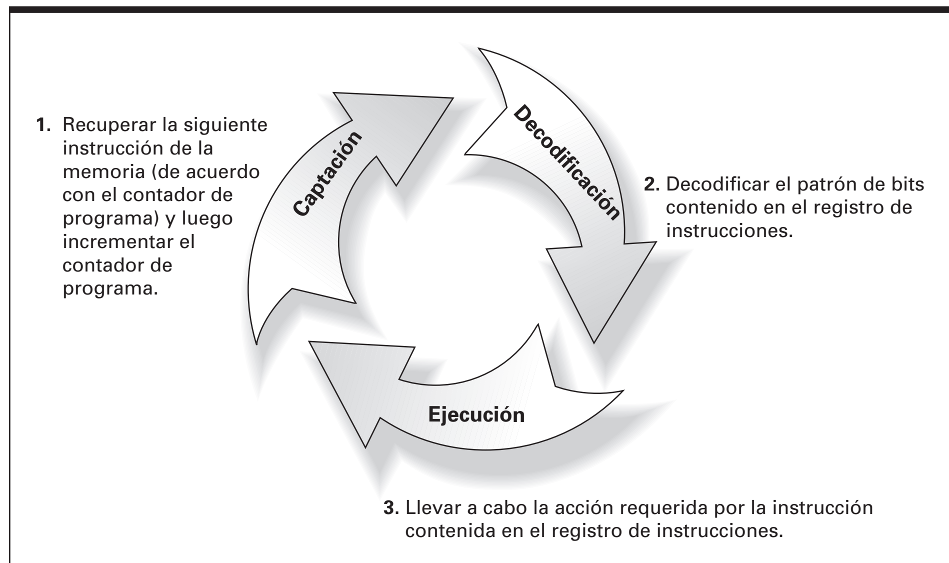
Para entender cómo tiene lugar el proceso global de ejecución, es necesario tener en cuenta dos de los registros de uso especial contenidos en el procesador: el **registro de instrucciones** y el **contador de programa** (véase de nuevo la Figura 2.4). El registro de instrucciones se utiliza para almacenar la instrucción que se está ejecutando. El contador de programa contiene la dirección de la siguiente instrucción que hay que ejecutar, por lo que sirve para que la máquina sepa en qué punto del programa se encuentra.

El procesador lleva a cabo su tarea repitiendo continuamente un algoritmo que le hace recorrer un proceso de tres pasos conocido con el nombre de **ciclo de máquina**. Los pasos del ciclo de máquina son la captación de instrucción, la decodificación y la ejecución (Figura 2.8). Durante el paso de captación, el procesador solicita que la memoria principal le proporcione la instrucción almacenada en la dirección indicada por el contador de programa. Puesto que cada instrucción de nuestra máquina tiene una longitud de dos bytes, este proceso de captación implica leer el contenido de dos celdas de la memoria principal. El procesador almacena la instrucción recibida de la memoria en su registro de instrucciones y luego incrementa el contador de programa en dos unidades, para que ese contador contenga la dirección de la siguiente instrucción almacenada en la memoria. De este modo, el contador de programa estará listo para la siguiente captación.

Teniendo ahora la instrucción en el registro de instrucciones, el procesador decodifica la instrucción, lo que implica descomponer el campo de operandos en sus correspondientes componentes, basándose en el código de operación de la instrucción.

El procesador ejecuta la instrucción activando la circuitería apropiada para llevar a cabo la tarea solicitada. Por ejemplo, si la instrucción es una carga desde la memoria, el procesador envía las señales apropiadas, espera a que la memoria principal envíe los datos y luego almacena esos datos en el registro solicitado; si la instrucción se corresponde con una operación aritmética, el procesador activa la circuitería apropiada de la unidad aritmético/lógica, utilizando los registros correctos como entradas y espera a que dicha unidad calcule la respuesta, después de lo cual la coloca en el registro apropiado.

Figura 2.8 Ciclo de máquina.



Una vez ejecutada la instrucción contenida en el registro de instrucciones, el procesador comienza un nuevo ciclo de máquina, ejecutando el paso correspondiente de captación. Observe que como el contador de programa fue incrementado al final de la captación anterior, de nuevo proporcionará al procesador la dirección correcta.

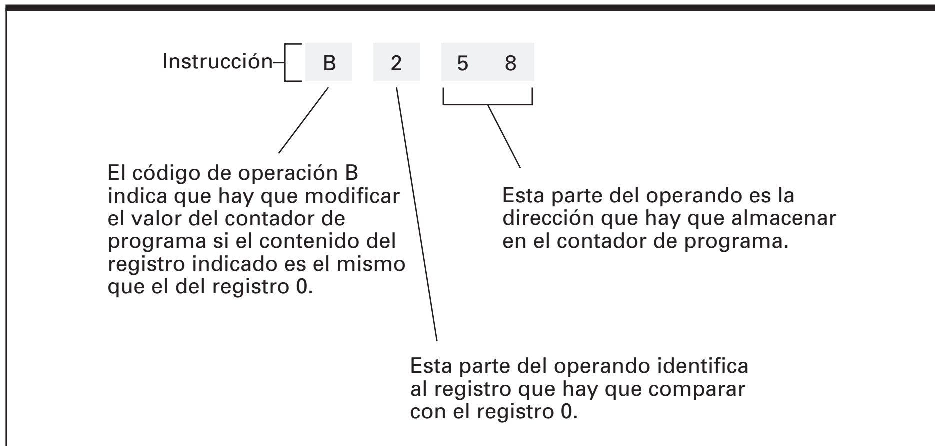
Un caso hasta cierto punto especial es la ejecución de una instrucción JUMP. Por ejemplo, considere la instrucción B258 (Figura 2.9), que significa "Saltar a la instrucción contenida en la dirección 58 (hexadecimal) si el contenido del registro 2 coincide con el contenido del registro 0". En este caso, el paso de ejecución del ciclo de máquina comienza con la comparación de los registros 2 y 0. Si contienen patrones de bits diferentes, el paso de ejecución termina y dará comienzo el siguiente ciclo de máquina. Sin embargo, si el contenido de los dos registros coincide, la máquina colocará el valor 58 (hexadecimal) en su contador de programa durante el paso de ejecución. En este caso, por tanto, la siguiente fase de captación se encontrará con el valor 58 en el contador de programa, por lo que la siguiente instrucción que habrá que extraer y ejecutar será la instrucción contenida en dicha dirección.

Observe que si la instrucción hubiera sido B058, entonces la decisión de si hay que cambiar el contador de programa dependería de que el contenido del registro 0 fuera igual al del registro 0. Pero como se trata del mismo registro, el contenido será siempre igual. Por ello, cualquier instrucción de la forma B0XY hará que se ejecute un salto a la dirección de memoria XY, independientemente de cuál sea el contenido del registro 0.

Ejemplo de ejecución de programa

Analicemos el ciclo de máquina aplicado al programa presentado en la Figura 2.7, que extrae dos valores de la memoria principal, calcula su suma y almacena

Figura 2.9 Decodificación de la instrucción B258.



el total en otra celda de la memoria principal. Primero necesitamos poner el programa en algún lugar de la memoria. Para nuestro ejemplo vamos a suponer que el programa se almacena en direcciones consecutivas, comenzando en la dirección A0 (hexadecimal). Con el programa almacenado de esta manera, podemos hacer que la máquina lo ejecute introduciendo en el contador de programa la dirección (A0) de la primera instrucción y haciendo que la máquina inicie las operaciones (Figura 2.10).

El procesador comienza el paso de captación del ciclo de máquina extrayendo la instrucción almacenada en la posición A0 de la memoria principal,

Comparación de la potencia de cálculo

A la hora de adquirir una computadora personal, se encontrará con que a veces se emplean las velocidades de reloj para comparar distintas máquinas. El **reloj** de una computadora es un circuito, denominado oscilador, que genera pulsos que se utilizan para coordinar las actividades de la máquina. Cuanto más rápido genere los pulsos este circuito de oscilación, más rápidamente podrá la máquina realizar su ciclo de máquina. Las velocidades de reloj suelen medirse en hercios (Hz) siendo un Hz igual a un ciclo (o pulso) por segundo. Las velocidades de reloj típicas de los equipos de sobremesa están en el rango que va de unos pocos centenares de MHz (en los modelos más antiguos) a varios GHz. (MHz significa megahercio, que es igual a un millón de Hz. GHz significa gigahercio y equivale a 1000 MHz.)

Lamentablemente, los diferentes diseños de procesador pueden realizar diferentes cantidades de trabajo útil en un ciclo de reloj, por lo que el utilizar únicamente la velocidad de reloj no es apropiado a la hora de comparar máquinas que tengan tipos de procesador distinto. Si estamos comparando una máquina basada en un procesador de Intel con otra basada en ARM, resultará más lógico comparar el rendimiento por medio de las denominadas **pruebas de rendimiento**, que son una forma de comparar el comportamiento de distintas máquinas al ejecutar un mismo programa. Seleccionando pruebas de rendimiento que representen distintos tipos de aplicaciones, podemos obtener comparaciones útiles para diversos segmentos de mercado.

colocando esta instrucción (156C) en su registro de instrucciones (Figura 2.11a). Observe que en nuestra máquina las instrucciones tienen 16 bits (dos bytes) de longitud. Por tanto, la instrucción completa que hay que extraer ocupa las celdas de memoria situadas en las direcciones A0 y A1. El procesador está diseñado para tener esto en cuenta, así que extrae el contenido de ambas celdas y coloca los patrones de bits recibidos en el registro de instrucciones, que tiene una longitud de 16 bits. El procesador suma entonces dos unidades al contador de programa para que el registro contenga la dirección de la siguiente instrucción (Figura 2.11b). Al final del paso de captación del primer ciclo de máquina, el contador del programa y el registro de instrucciones contendrán los datos siguientes:

```
Contador de programa: A2
Registro de instrucciones: 156C
```

A continuación, el procesador analiza la instrucción contenida en su registro de instrucciones y concluye que se necesita cargar el registro 5 con el contenido de la celda de memoria situada en la dirección 6C. Esta actividad de carga se realiza durante el paso de ejecución del ciclo de máquina, después de lo cual el procesador inicia el siguiente ciclo.

Este segundo ciclo comienza extrayendo la instrucción 166D de las dos celdas de memoria que comienzan en la dirección A2. El procesador coloca esta instrucción en el registro de instrucciones e incrementa el contador de programa al valor A4. Los valores del contador de programa y del registro de instrucciones serán entonces los siguientes:

Figura 2.10 El programa de la Figura 2.7 almacenado en la memoria principal preparado para su ejecución.

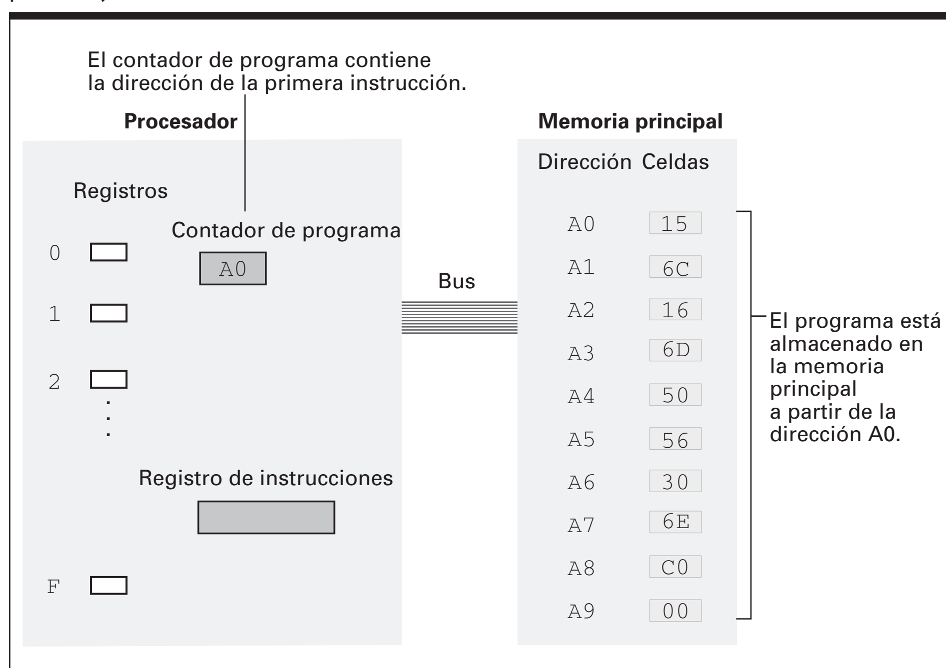
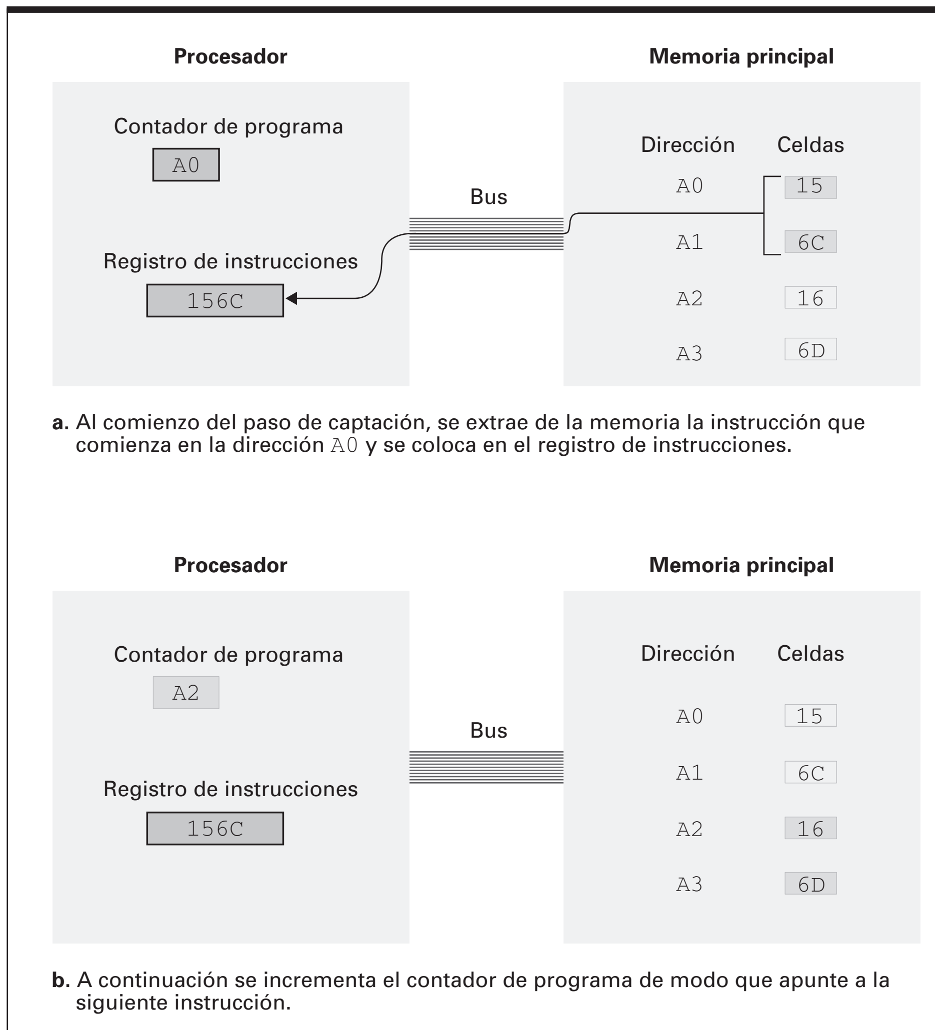


Figura 2.11 Paso de captación del ciclo de máquina.



a. Al comienzo del paso de captación, se extrae de la memoria la instrucción que comienza en la dirección `A0` y se coloca en el registro de instrucciones.

b. A continuación se incrementa el contador de programa de modo que apunte a la siguiente instrucción.

```
Contador de programa: A4
Registro de instrucciones: 166D
```

Ahora el procesador decodifica la instrucción 166D y determina que se necesita cargar el contenido del registro 6 con el contenido de la dirección de memoria 6D. A continuación se ejecuta la instrucción. Es en este momento cuando se carga realmente el registro 6.

Puesto que el contador de programa contiene el valor A4, el procesador extrae la siguiente instrucción comenzando en dicha dirección. El resultado es que se coloca el valor 5056 en el registro de instrucciones y se incrementa el contador del programa al valor A6. El procesador decodifica ahora el contenido de su registro de instrucciones y lo ejecuta, activando la circuitería de suma en complemento a dos, utilizando como entradas los registros 5 y 6.

Durante este paso de ejecución, la unidad aritmético/lógica realiza la suma solicitada, almacena el resultado en el registro 0 (tal como le ha indicado la uni-

dad de control) e informa a la unidad de control de que ha terminado. El procesador inicia entonces otro ciclo de máquina. Una vez más, con la ayuda del contador de programa, extrae la siguiente instrucción (306E) de las dos celdas de memoria que comienzan en la dirección A6 e incrementa el contador de programa al valor A8. A continuación se decodifica esta instrucción y se ejecuta. En este punto, se coloca la suma en la posición de memoria 6E.

La siguiente instrucción se extrae comenzando en la posición de memoria A8 y el contador de programa se incrementa al valor AA. A continuación se decodifica el contenido del registro de instrucciones (C000) viéndose que se corresponde con la instrucción de detención. En consecuencia, la máquina se detiene durante el paso de ejecución del ciclo de máquina, con lo que el programa se completa.

En resumen, vemos que la ejecución de un programa almacenado en la memoria implica el mismo tipo de proceso que cualquiera de nosotros usaría al seguir una lista de instrucciones detallada. Mientras que nosotros podemos saber en qué punto de la lista nos encontramos tachando las instrucciones a medida que las ejecutamos, el procesador sabe dónde se encuentra mediante el contador de programa. Después de determinar qué instrucción hay que ejecutar a continuación, deberíamos leer la instrucción y trataríamos de comprender su significado. Después, realizaríamos la tarea solicitada y volveríamos a consultar la lista para ver cuál es la siguiente instrucción, de la misma manera que el procesador ejecuta la instrucción contenida en su registro de instrucciones y luego continúa con otra captación.

Programas y datos

En la memoria principal de una computadora podemos almacenar simultáneamente muchos programas, siempre y cuando ocupen partes diferentes de la memoria. Después, podemos determinar qué programa se ejecutará al iniciar la máquina simplemente configurando de la forma apropiada el contador de programa.

Sin embargo, debemos tener en mente que puesto que la memoria principal contiene también datos y esos datos están codificados mediante 0s y 1s, la máquina no tiene manera por si sola de saber qué patrones de bits se corresponden con datos y cuáles se corresponden con programas. Si asignáramos al contador de programa la dirección de una parte de la memoria que contiene datos, en lugar de la dirección del programa deseado, el procesador, al no disponer de ninguna otra información, extraería los patrones de bits correspondientes a los datos como si fueran instrucciones y los ejecutaría. El resultado final dependería por supuesto del valor de esos datos.

De esto no debemos deducir, sin embargo, que sea malo proporcionar programas y datos a la memoria de una máquina con una apariencia común. De hecho, es una característica que en ocasiones puede ser útil, porque permite a un programa manipular otros programas (o incluso manipularse a sí mismo) de la misma manera que manipularía datos normales. Por ejemplo, imagine un programa que se modificara a sí mismo en respuesta a las interacciones con el entorno y que exhibiera la capacidad de aprender. O imagine un programa que escribiera y ejecutara otros programas para tratar de resolver los problemas que le planteáramos.

Cuestiones y ejercicios

1. Suponga que las celdas de memoria de las direcciones 00 a 05 de la máquina descrita en el Apéndice C contienen los patrones de bits (hexadecimales) dados en la siguiente tabla:

Dirección	Contenido
00	14
01	02
02	34
03	17
04	C0
05	00

Si iniciamos la máquina teniendo el contador de programa el valor 00, ¿que patrón de bits estará almacenado en la celda de memoria cuya dirección es el valor hexadecimal 17 en el momento de detenerse la máquina?

2. Suponga que las celdas de memoria de las direcciones B0 a B8 de la máquina descrita en el Apéndice C contienen los patrones de bits (hexadecimales) dados en la siguiente tabla:

Dirección	Contenido
B0	13
B1	B8
B2	A3
B3	02
B4	33
B5	B8
B6	C0
B7	00
B8	0F

a. Si el contador de programa comienza en B0, ¿qué patrón de bits contendrá el registro número 3 después de ejecutar la primera instrucción?

b. ¿Qué patrón de bits contendrá la celda B8 cuando se ejecute la instrucción de detención?

3. Suponga que las celdas de memoria de las direcciones A4 a B1 de la máquina descrita en el Apéndice C contienen los patrones de bits (hexadecimales) dados en la primera tabla de la página siguiente.

A la hora de responder a las siguientes cuestiones, suponga que iniciamos la máquina con el valor A4 en el contador de programa.

a. ¿Qué contendrá el registro 0 la primera vez que se ejecuta la instrucción contenida en la dirección AA?

b. ¿Qué contendrá el registro 0 la segunda vez que se ejecuta la instrucción contenida en la dirección AA?

c. ¿Cuántas veces se ejecuta la instrucción contenida en la dirección AA antes de que la máquina se detenga?

Dirección	Contenido
A4	20
A5	00
A6	21
A7	03
A8	22
A9	01
AA	B1
AB	B0
AC	50
AD	02
AE	B0
AF	AA
B0	C0
B1	00

4. Suponga que las celdas de memoria de las direcciones F0 a F9 de la máquina descrita en el Apéndice C contienen los patrones de bits (hexadecimales) dados en la siguiente tabla:

Dirección	Contenido
F0	20
F1	C0
F2	30
F3	F8
F4	20
F5	00
F6	30
F7	F9
F8	FF
F9	FF

Si la máquina inicia sus operaciones teniendo el valor F0 en el contador de programa, ¿qué hará la máquina cuando alcance la instrucción contenida en la dirección F8?

2.4 Instrucciones aritmético/lógicas

Como hemos indicado anteriormente, el grupo de instrucciones aritmético/lógicas está compuesto por instrucciones que codifican operaciones aritméticas, lógicas y de desplazamiento. En esta sección, vamos a examinar estas operaciones con más detalle.

Operaciones lógicas

En el Capítulo 1 hemos presentado las operaciones lógicas AND, OR y XOR (OR exclusiva) como operaciones que combinan dos bits de entrada para generar un único bit de salida. Estas operaciones pueden ampliarse a otras operaciones que combinen dos cadenas de bits para generar una única cadena de salida, aplicando la operación básica a todas las columnas individuales. Por ejemplo, el resultado de combinar mediante AND los patrones 10011010 y 11001001 es

```
    10011010
AND 11001001
    10001000
```

donde simplemente hemos escrito, debajo de cada columna, el resultado de combinar mediante AND los 2 bits de dicha columna. De la misma forma, la combinación mediante OR y XOR de estos patrones sería

```
   10011010            10011010
OR 11001001        XOR 11001001
   11011011            01010011
```

Uno de los principales usos de la operación AND es colocar una serie de ceros en una parte de un patrón de bits, sin afectar al valor contenido en el resto del patrón. Por ejemplo, considere lo que sucede si el byte 00001111 es el primer operando de una operación AND. Sin necesidad de conocer el contenido del segundo operando, podemos concluir que los cuatro bits más significativos del resultado serán igual a 0. Además, también podemos deducir que los cuatro bits menos significativos del resultado serán una copia de la parte correspondiente del segundo operando, como se muestra en el siguiente ejemplo:

```
    00001111
AND 10101010
    00001010
```

Este uso de la operación AND es un ejemplo del proceso conocido como **enmascaramiento**. En él, un operando denominado **máscara**, determina qué parte del otro operando afectará al resultado. En este caso de la operación AND, el enmascaramiento produce un resultado que es una réplica parcial de uno de los operandos, con una serie de 0s ocupando las posiciones no duplicadas.

Dicha operación es útil a la hora de manipular un **mapa de bits**, una cadena de bits en la que cada bit representa la presencia o ausencia de un objeto concreto. Ya nos hemos encontrado con los mapas de bits en el contexto de la representación de imágenes, donde cada bit estaba asociado con un píxel. Veamos otros ejemplos. Una cadena de 52 bits, en la que cada bit está asociado con una carta concreta de la baraja francesa; dicha cadena puede utilizarse para representar una mano de póker, asignando 1s a los cinco bits asociados con las cartas que tenemos en la mano y 0s a todos los bits restantes. De la misma forma, un mapa de bits de 52 bits, de los cuales trece sean igual a 1, puede utilizarse para representar una mano de bridge, mientras que un mapa de bits de 32 bits se puede emplear para representar qué helados de entre treinta y dos posibles sabores están disponibles.

Suponga entonces que utilizamos los 8 bits de una celda de memoria como mapa de bits y que queremos averiguar si está presente el objeto asociado con el tercer bit comenzando por el extremo de mayor peso. Lo único que tenemos que hacer es combinar mediante AND el byte completo con la máscara 00100000, lo que generará un byte compuesto únicamente por 0s si y solo si el tercer bit contando a partir del extremo de mayor peso del mapa de bits es también un 0. Un programa podría entonces actuar correspondientemente, incluyendo a continuación de la operación AND una instrucción de bifurcación condicional. Además, si el tercer bit a partir del extremo de mayor peso del mapa de bits es un 1 y queremos modificarlo para que tenga el valor 0 sin perturbar a los bits restantes, podemos combinar mediante AND el mapa de bits con la máscara 11011111 y luego almacenar el resultado en lugar del mapa de bits original.

Mientras que la operación AND puede utilizarse para duplicar una parte de una cadena de bits al mismo tiempo que se almacenan 0s en la parte no duplicada, la operación OR puede emplearse para duplicar una parte de una cadena al mismo tiempo que se almacenan 1s en la parte no duplicada. Para esto, de nuevo empleamos una máscara, pero esta vez indicamos las posiciones de bit que queremos duplicar mediante 0s, utilizando los 1s para indicar las posiciones no duplicadas. Por ejemplo, si combinamos cualquier byte mediante OR con la máscara 11110000 obtendremos un resultado que tendrá 1s en los cuatro bits más significativos, mientras que los bits restantes serán una copia de los cuatro bits menos significativos del otro operando, como se ilustra en el siguiente ejemplo:

```
   11110000
OR 10101010
   --------
   11111010
```

En consecuencia, mientras que la máscara 11011111 puede utilizarse con la operación AND para forzar a que el tercer bit a partir del extremo de mayor de peso de un byte tenga un valor igual a 0, la máscara 00100000 puede usarse con la operación OR para forzar que haya un 1 en dicha posición.

Uno de los usos principales de la operación XOR es generar el complemento de una cadena de bits. Si combinamos un byte mediante XOR con una máscara formada por todo 1s, obtenemos el complemento de dicho byte. Por ejemplo, observe la relación entre el segundo operando y el resultado en el siguiente ejemplo:

```
    11111111
XOR 10101010
    --------
    01010101
```

En el lenguaje máquina descrito en el Apéndice C, los códigos de operación 7, 8 y 9 se utilizan para las operaciones lógicas OR, AND y XOR, respectivamente. Cada una de estas instrucciones solicita que se lleve a cabo la correspondiente operación lógica entre los contenidos de dos registros designados y que se coloque el resultado en otro registro designado. Por ejemplo, la instrucción 7ABC solicita que se almacene en el registro A el resultado de combinar mediante OR el contenido de los registros B y C.

Operaciones de rotación y desplazamiento

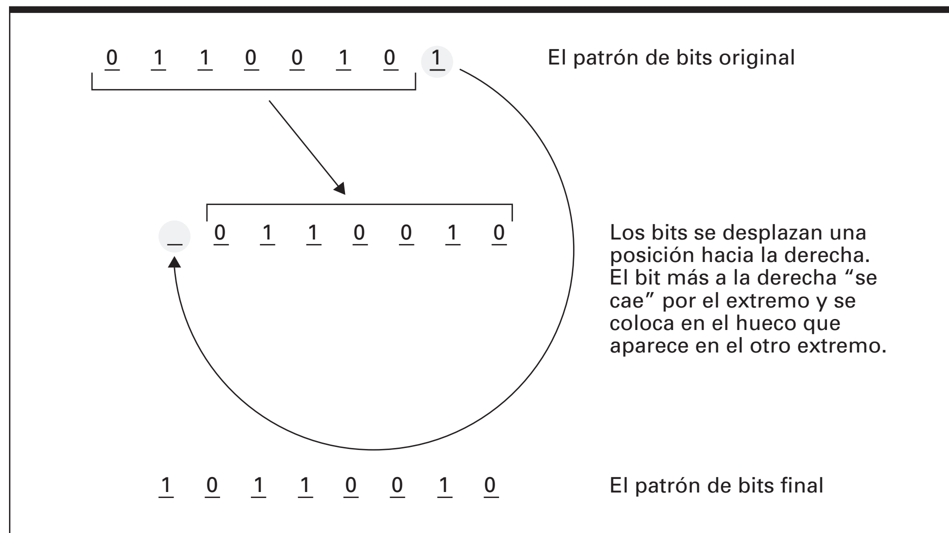
Las operaciones pertenecientes a la clase de operaciones de rotación y desplazamiento proporcionan un medio para mover los bits dentro de un registro y se utilizan a menudo para resolver problemas de alineamiento. Estas operaciones se clasifican por la dirección del movimiento (hacia la derecha o hacia la izquierda) y según que el proceso sea circular o no. Con estas directrices de clasificación existen numerosas variantes, que tienen una terminología mixta. Examinemos brevemente los principales conceptos.

Considere un registro que contenga un byte de bits. Si desplazamos su contenido 1 bit hacia la derecha podemos imaginarnos al bit de más a la derecha cayéndose por el borde, al mismo tiempo que aparece un agujero en el extremo izquierdo del byte. Lo que suceda con este bit sobrante y con el agujero es lo que permite distinguir entre las distintas operaciones de desplazamiento. Una técnica consiste en colocar en el agujero que aparece en el extremo izquierdo el bit que se ha caído por el extremo derecho. El resultado es un **desplazamiento circular**, también denominado **rotación**. Por tanto, si realizamos ocho veces consecutivas un desplazamiento circular a la derecha de un patrón de bits de un byte de tamaño, obtenemos el mismo patrón de bits con el que comenzamos.

Otra técnica consiste en descartar el bit que se ha caído por el borde y rellenar el agujero con un 0. Para hacer referencia a estas operaciones suele utilizarse el término **desplazamiento lógico**. Dichos desplazamientos hacia la izquierda normalmente se emplean para multiplicar por dos los números con representación de complemento a dos. Después de todo, el desplazamiento de los dígitos binarios hacia la izquierda se corresponde con una multiplicación por dos, de la misma forma que un desplazamiento similar de una cadena de dígitos decimales se corresponde con una multiplicación por diez. Asimismo, puede realizarse la división por dos desplazando la cadena binaria hacia la derecha. En cada desplazamiento, es preciso tener cuidado para preservar el bit de signo, cuando se emplean ciertos sistemas de notación. Por ello, a menudo podemos encontrar desplazamientos a la derecha que siempre rellenan el agujero (que se produce en la posición correspondiente al bit de signo) con su valor original. Los desplazamientos que no modifican el bit de signo se denominan en ocasiones **desplazamientos aritméticos**.

Entre las distintas instrucciones posibles de rotación y desplazamiento, el lenguaje máquina descrito en el Apéndice C solo contiene un desplazamiento circular a la derecha, designado mediante el código de operación A. En este caso, el primer dígito hexadecimal del operando especifica el registro que hay que rotar y el resto del operando especifica el número de bits que hay que rotar. Por tanto, la instrucción A501 significa "Rotar el contenido del registro 5 un bit a la derecha". En particular, si originalmente el registro 5 contenía el patrón de bits 65 (hexadecimal), entonces contendrá B2 después de ejecutar esta instrucción (Figura 2.12). (Trate de experimentar con el modo en que pueden realizarse otras instrucciones de desplazamiento y rotación mediante combinaciones de las instrucciones proporcionadas en el lenguaje máquina del Apéndice C. Por ejemplo, puesto que un registro tiene 8 bits de longitud, un desplazamiento circular hacia la derecha de 3 bits produce el mismo resultado que un desplazamiento circular de 5 bits hacia la izquierda.)

Figura 2.12 Rotación del patrón de bits 65 (hexadecimal) un bit hacia la derecha.



Operaciones aritméticas

Aunque ya hemos hablado de las operaciones aritméticas de suma, resta, multiplicación y división, nos quedan unos cuantos cabos sueltos. En primer lugar, ya hemos visto que la resta puede simularse mediante las operaciones de suma y negación. Además, la multiplicación es simplemente una suma repetida, mientras que la división es una resta repetida (seis dividido entre dos es tres porque podemos restar de seis tres veces dos). Por esta razón, algunos procesadores pequeños se diseñan con solo la instrucción de suma, o quizá solo la de suma y la de resta.

También es preciso mencionar que existen numerosas variantes de cada operación aritmética. Ya hemos aludido a esto en relación con las operaciones de suma disponibles en nuestra máquina descrita en el Apéndice C. En el caso de la suma, por ejemplo, si los valores que hay que sumar están almacenados en notación de complemento a dos, el proceso de suma debe realizarse mediante una suma directa columna a columna. Sin embargo, si los operandos están almacenados como valores en punto flotante, el proceso de suma debe extraer la mantisa de cada uno, desplazarlos hacia la izquierda o hacia la derecha de acuerdo con los campos de exponente, comprobar los bits de signo, efectuar la suma y traducir el resultado a notación en punto flotante. Por tanto, aunque ambas operaciones se consideran una suma, las acciones que lleva a cabo la máquina en cada caso no son las mismas.

Cuestiones y ejercicios

1. Realice las siguientes operaciones.

a.
```
    01001011
AND 10101011
```

b.
```
    10000011
AND 11101100
```

c.
```
    11111111
AND 00101101
```

d. 01001011 OR 10101011	**e.** 10000011 OR 11101100	**f.** 11111111 OR 00101101
g. 01001011 XOR 10101011	**h.** 10000011 XOR 11101100	**i.** 11111111 XOR 00101101

2. Suponga que desea aislar los 4 bits centrales de un byte, colocando 0s en los otros 4 bits pero sin afectar a los bits centrales. ¿Qué máscara debería utilizar y con qué operación?

3. Suponga que desea complementar los 4 bits centrales de un byte, sin perturbar a los otros 4 bits. ¿Qué máscara debería utilizar y con qué operación?

4. **a.** Suponga que combina mediante XOR los 2 primeros bits de una cadena de bits y luego continúa recorriendo la cadena, combinando sucesivamente mediante XOR cada resultado con el siguiente bit de la cadena. ¿Cómo se relaciona el resultado con el número de 1s que aparece en la cadena?

b. ¿Cómo se relaciona este problema con el proceso de determinación de cuál debería ser el bit de paridad apropiado a la hora de codificar un mensaje?

5. A menudo es conveniente utilizar una operación lógica en lugar de una numérica. Por ejemplo, la operación lógica AND combina 2 bits de la misma forma que la multiplicación. ¿Qué operación lógica es casi igual que la suma de 2 bits y qué es lo que falla en este caso?

6. ¿Qué operación lógica y qué máscara podrían emplearse para cambiar los códigos ASCII de las letras minúsculas a mayúsculas? ¿Y para cambiar los de las mayúsculas a minúsculas?

7. ¿Cuál es el resultado de realizar un desplazamiento circular de 3 bits hacia la derecha en las siguientes cadenas de bits:

a. 01101010 **b.** 00001111 **c.** 01111111

8. ¿Cuál es el resultado de realizar un desplazamiento circular de 1 bit hacia la izquierda en los siguientes bytes representados en notación hexadecimal? Proporcione la respuesta en formato hexadecimal.

a. AB **b.** 5C **c.** B7 **d.** 35

9. ¿A cuántos bits de desplazamiento circular hacia la izquierda equivale un desplazamiento circular hacia la derecha de 3 bits si estamos trabajando con una cadena de 8 bits?

10. ¿Qué patrón de bits representa la suma de 01101010 y 11001100 si los patrones representan valores almacenados en notación de complemento a dos? ¿Y en el caso de que los patrones representen valores almacenados en el formato de punto flotante explicado en el Capítulo 1?

11. Utilizando el lenguaje máquina del Apéndice C, escriba un programa que almacene un 1 en el bit más significativo de la celda de memoria cuya dirección es A7 sin modificar los restantes bits de la celda.

12. Usando el lenguaje máquina del Apéndice C, escriba un programa que copie los 4 bits centrales de la celda de memoria E0 en los 4 bits menos significativos de la celda de memoria E1, al mismo tiempo que se colocan 0s en los 4 bits más significativos de la celda situada en la posición E1.

2.5 Comunicación con otros dispositivos

La memoria principal y el procesador forman el núcleo de una computadora. En esta sección vamos a ver cómo este núcleo, al que denominaremos computadora, se comunica con los dispositivos periféricos, como por ejemplo los sistemas de almacenamiento masivo, las impresoras, teclados, ratones, pantallas, cámaras digitales e incluso otras computadoras.

El papel de las controladoras

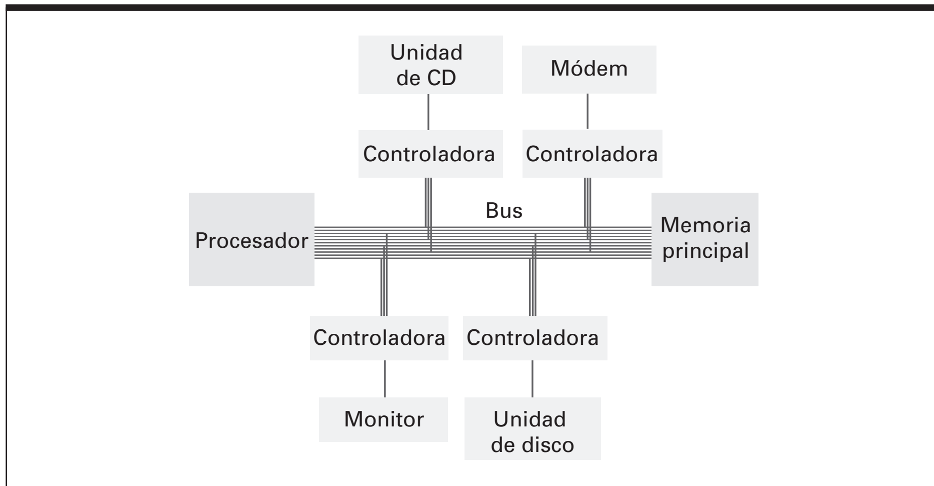
La comunicación entre una computadora y otros dispositivos suele gestionarse mediante un aparato intermedio denominado **controladora**. En el caso de una computadora personal, una controladora puede constar de circuitos montados de forma permanente en la placa base de la computadora o, para disponer de una mayor flexibilidad, puede adoptar la forma de una tarjeta de circuito que se inserta en una de las ranuras de la placa madre. En cualquiera de los dos casos, la controladora se conecta mediante cables a dispositivos periféricos situados dentro de la carcasa de la computadora o quizá a un conector, denominado **puerto**, en la parte posterior de la computadora, al que pueden conectarse dispositivos externos. Estas controladoras son en ocasiones pequeñas computadoras en sí mismas, cada una con su propia circuitería de memoria y con un procesador simple que ejecuta un programa encargado de dirigir las actividades de la controladora.

Una controladora traduce los mensajes y los datos entre un formato compatible con las características internas de la computadora y el formato compatible con el dispositivo periférico al que está conectada. Originalmente, cada controladora se diseñaba para un tipo concreto de dispositivo; por tanto, la adquisición de un nuevo dispositivo periférico requería a menudo que se adquiriera también una nueva controladora.

Recientemente, se han dado pasos dentro del campo de las computadoras personales para desarrollar una serie de estándares, como el **bus serie universal** (USB, *Universal Serial Bus*) y **FireWire**, que permiten que una misma controladora sea capaz de gestionar diversos tipos de dispositivos. Por ejemplo, una única controladora USB puede emplearse como interfaz entre una computadora y cualquier conjunto de dispositivos compatibles con USB. La lista de dispositivos existentes hoy día en el mercado y que pueden comunicarse con una controladora USB incluye ratones, impresoras, escáneres, dispositivos de almacenamiento masivo, cámaras digitales y teléfonos inteligentes.

Cada controladora se comunica con la propia computadora por medio de una serie de conexiones al mismo bus que interconecta el procesador y la memoria principal de la computadora (Figura 2.13). Desde esta posición, la con-

Figura 2.13 Controladoras conectadas al bus de una máquina.



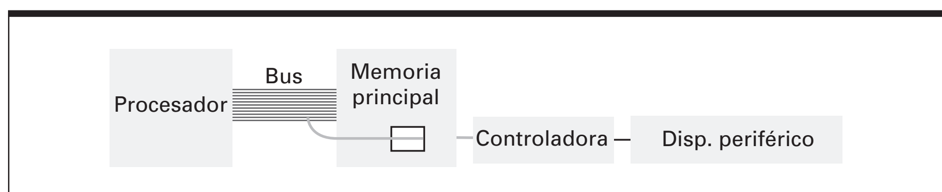
troladora es capaz de monitorizar las señales que intercambian el procesador y la memoria principal, así como inyectar sus propias señales en el bus.

Con esta disposición, el procesador es capaz de comunicarse con las controladoras conectadas al bus, de la misma forma que se comunica con la memoria principal. Para enviar un patrón de bits a una controladora, primero se construye dicho patrón en uno de los registros de uso general del procesador. Después, el procesador ejecuta una instrucción similar a la instrucción STORE para "almacenar" el patrón de bits en la controladora. De la misma forma, para recibir un patrón de bits desde una controladora, se utiliza una instrucción similar a la instrucción LOAD.

En algunos diseños de computadoras, la transferencia de datos hacia y desde las controladoras se efectúa con los mismos códigos de operación LOAD y STORE que se utilizan para la comunicación con la memoria principal. En esos casos, cada controladora está diseñada para responder a las referencias dirigidas a un conjunto distintivo de direcciones, mientras que la memoria principal está diseñada para ignorar las referencias a dichas ubicaciones. De este modo, cuando el procesador envía un mensaje a través del bus para almacenar un patrón de bits en una ubicación de memoria asignada a una controladora, el patrón de bits se "almacena" en realidad en la controladora en lugar de en la memoria principal. De la misma forma, si el procesador trata de leer datos de una de esas posiciones de memoria, como por ejemplo en una instrucción LOAD, recibirá un patrón de bits de la controladora en lugar de recibirlo de la memoria. Este tipo de sistema de comunicación se denomina **E/S mapeada en memoria** porque los dispositivos de entrada/salida de la computadora parecen estar en diversas posiciones de memoria (Figura 2.14).

Una alternativa a la E/S mapeada en memoria consiste en proporcionar códigos de operación especiales dentro del lenguaje máquina para dirigir las transferencias hacia y desde las controladoras. Las instrucciones con dichos códigos de operación se denominan instrucciones de E/S. Por ejemplo, si el lenguaje descrito en el Apéndice C utilizara este método, podría incluir una ins-

Figura 2.14 Representación conceptual de la E/S mapeada en memoria.



trucción tal como F5A3, que significara "Almacenar el contenido del registro 5 en la controladora identificada por el patrón de bits A3".

Acceso directo a memoria

Puesto que una controladora está conectada al bus de la computadora, puede efectuar sus propias comunicaciones con la memoria principal durante aquellos nanosegundos durante los que el procesador no está utilizando el bus. Esta capacidad de una controladora para acceder a la memoria principal se conoce con el nombre de **acceso directo a memoria** (DMA, *Direct Memory Access*), y es una característica de gran importancia para mejorar el rendimiento de una computadora. Por ejemplo, para extraer datos de un sector de un disco, el procesador puede enviar solicitudes codificadas como patrones de bits a la controladora asociada al disco, pidiéndole que lea el sector y coloque los datos en un área especificada de la memoria principal. El procesador puede entonces continuar con otras tareas mientras la controladora lleva a cabo la operación de lec-

USB y FireWire

El bus serie universal (USB) y FireWire son sistemas de comunicación serie estandarizados que simplifican el proceso de añadir nuevos dispositivos periféricos a una computadora personal. USB fue desarrollado bajo la dirección de Intel. El desarrollo de FireWire fue liderado por Apple. En ambos casos, el objetivo último es que una única controladora pueda proporcionar puertos externos a los que se puedan conectar diversos dispositivos periféricos. Con este tipo de solución, la controladora traduce las características internas de las señales de la computadora a las señales estándar para USB o FireWire. A su vez, cada dispositivo conectado a la controladora convierte sus detalles internos de operación al mismo estándar USB o FireWire, permitiéndose así la comunicación con la controladora. El resultado es que conectar un nuevo dispositivo a un PC no requiere la inserción de una nueva controladora. En lugar de ello, basta con enchufar en un puerto USB cualquier dispositivo compatible con USB o conectar a un puerto FireWire cualquier dispositivo compatible con FireWire.

De los dos estándares, FireWire es el que proporciona una tasa de transferencia más rápida, pero el menor coste de la tecnología USB la ha convertido en la solución líder dentro del sector del mercado de masas, donde el precio es más crítico. Entre los dispositivos compatibles con USB que podemos encontrar hoy día en el mercado tenemos ratones, teclados, impresoras, escáneres, cámaras digitales, teléfonos inteligentes y sistemas de almacenamiento masivo diseñados para aplicaciones de archivo. Las aplicaciones FireWire tienden a centrarse en dispositivos que requieren tasas de transferencia mayores, como grabadoras de vídeo y sistemas de almacenamiento masivo en línea.

tura y deposita los datos en la memoria principal mediante el mecanismo de DMA. Así, la computadora realizará dos actividades a un mismo tiempo. El procesador estará ejecutando un programa y la controladora estará dirigiendo la transferencia de datos entre el disco y la memoria principal. De esta forma, los recursos de computación del procesador no se desperdician durante la transferencia de datos, que es relativamente lenta.

La utilización de DMA tiene también una desventaja, que es que complica las comunicaciones que tienen lugar a través del bus de una computadora. Es necesario transferir patrones de bits entre el procesador y la memoria principal, entre el procesador y cada controladora y entre cada controladora y la memoria principal. La coordinación de todas estas actividades que tienen lugar en el bus constituyen un problema de diseño significativo. Incluso con diseños excelentes, el bus central puede transformarse en un impedimento, a medida que el procesador y las controladoras compiten por el acceso al bus. Este impedimento se conoce con el nombre de **cuello de botella de von Neumann**, porque es una consecuencia de la **arquitectura de von Neumann** subyacente, en la que el procesador extrae sus instrucciones de la memoria a través de un bus central.

Handshaking

La transferencia de datos entre dos componentes de una computadora raramente suele ser de tipo unidireccional. Aunque podemos pensar en una impresora como en un dispositivo que solo recibe datos, la verdad es que una impresora también devuelve datos a la computadora. Después de todo, una computadora puede generar y enviar caracteres a una impresora mucho más rápido de lo que la impresora los puede imprimir. Si la computadora enviara ciegamente los datos a una impresora, esta quedaría rezagada rápidamente, lo que provocaría la pérdida de datos. Por tanto, un proceso tal como la impresión de un documento implica un diálogo bidireccional constante, conocido como proceso de **coordinación** (*handshaking*) en el que la computadora y el dispositivo periférico intercambian información acerca del estado del dispositivo y coordinan sus actividades.

Este proceso de coordinación implica a menudo el uso de una **palabra de estado**, que es un patrón de bits generado por el dispositivo periférico y que se envía a la controladora. La palabra de estado es un mapa de bits en el que cada bit refleja una determinada condición del dispositivo. Por ejemplo, en el caso de una impresora, el valor del bit menos significativo de la palabra de estado podría indicar si la impresora se ha quedado sin papel, mientras que el siguiente bit podría indicar si la impresora está lista para recibir datos adicionales. Algún otro bit podría emplearse para indicar que se ha producido un atasco de papel. Dependiendo del sistema, la controladora puede responder a dicha información de estado por sí misma o poner esa información a disposición del procesador. En cualquier caso, la palabra de estado proporciona el mecanismo para poder coordinar la comunicación con un dispositivo periférico.

Medios de comunicación populares

La comunicación entre dispositivos de computación tiene lugar a través de dos tipos de enlaces: paralelos y serie. Estos términos hacen referencia a la forma

en que se transfieren unas señales con respecto a otras. En el caso de la **comunicación paralelo**, varias señales se transfieren al mismo tiempo a través de una "línea" separada. Dicha técnica es capaz de transferir los datos rápidamente pero requiere una ruta de comunicaciones relativamente compleja. Como ejemplo podemos citar el bus interno de una computadora, en el que se emplean múltiples hilos para poder transferir simultáneamente grandes bloques de datos y de otras señales.

Por el contrario, la **comunicación serie** se basa en transferir señales una detrás de otra a través de una única línea. Por tanto, la comunicación serie requiere una ruta de datos más simple que la comunicación paralelo, lo cual es la razón de su popularidad. USB y FireWire, que ofrecen una transferencia de datos de velocidad relativamente alta a cortas distancia (de solo unos cuantos metros) son ejemplos de sistemas de comunicación serie. Para distancias ligeramente mayores (dentro de una vivienda o de un edificio de oficinas), es muy común utilizar comunicación serie a través de conexiones Ethernet (Sección 4.1), bien mediante cable o difusión por radio.

Para comunicación a distancias mayores, las líneas telefónicas tradicionales han dominado el campo de las computadoras personales durante muchos años. Estas rutas de comunicación compuestas por un único cable a través del cual se transfieren tonos, uno detrás de otro, son inherentemente sistemas serie. La transferencia de datos digitales a través de estas líneas se realiza convirtiendo primero los patrones de bits en tonos audibles por medio de un **módem** (abreviatura de ***modulador-demodulador***), transfiriendo estos tonos en forma serie a través del sistema telefónico y luego convirtiendo los tonos a bits, mediante otro módem situado en el destino.

Para comunicaciones a larga distancia sobre líneas telefónicas tradicionales, las compañías telefónicas ofrecen un servicio conocido con el nombre de **DSL** (***Digital Subscriber Line,*** Línea digital de abonado), que aprovecha el hecho de que las líneas telefónicas existentes pueden admitir un rango de frecuencias mayor que el usado por las comunicaciones de voz tradicionales. Para ser más precisos, DSL utiliza las frecuencias por encima del rango audible para transferir datos digitales, mientras que deja el espectro de frecuencias más bajo para las comunicaciones de voz. Aunque DSL ha tenido un gran éxito, las compañías telefónicas están actualizando rápidamente sus sistemas a líneas de fibra óptica, que soportan las comunicaciones digitales con mayor fiabilidad que las líneas telefónicas tradicionales.

Otras tecnologías que compiten con DSL y la fibra óptica incluyen el cable, como el utilizado en los sistemas de televisión por cable y los enlaces vía satélite, que emplean difusiones de radio de alta frecuencia.

Velocidades de comunicación

La velocidad con que se transfieren bits de un componente de una computadora a otro se mide en **bits por segundo (bps)**. Entre las unidades más comunes podemos citar el **Kbps** (kilo-bps, que es igual a mil bps), el **Mbps** (mega-bps, igual a un millón de bps) y el **Gbps** (giga-bps, igual a mil millones de bps). (Observe la distinción entre bits y bytes; es decir, 8 Kbps es igual a 1 KB por segundo. Cuando se emplean abreviaturas, una b minúscula suele hacer referencia a ***bit*** mientras que una B mayúscula significa ***byte***.)

Para comunicaciones a corta distancia, USB y FireWire proporcionan tasas de transferencia de varios cientos de Mbps, lo que es suficiente para la mayoría de las aplicaciones multimedia. Esto, combinado con la comodidad que proporcionan y su coste relativamente bajo, es la razón de que hayan ganado tanta popularidad como sistemas de comunicación entre computadoras domésticas y periféricos locales como por ejemplo impresoras, unidades de disco externas y cámaras.

Combinando las técnicas de **multiplexación** (la codificación o entrelazado de datos, de modo que un único enlace de comunicaciones actúe como si fuera un conjunto de múltiples enlaces) y de compresión de datos, los sistemas telefónicos de voz tradicionales eran capaces de soportar tasas de transferencia de 57,6 Kbps, lo que resulta demasiado escaso para las necesidades de las aplicaciones multimedia de Internet actuales, como YouTube y Facebook. Para reproducir grabaciones de música codificadas con MP3 se necesita una tasa de transferencia de aproximadamente 64 Kbps y para reproducir videoclips incluso de muy baja calidad hacen falta tasas de transferencia en el rango de los Mbps. Esa es la razón de que otras alternativas como DSL, cable y enlaces vía satélite, que proporcionan tasas de transferencia en el rango de los Mbps, hayan sustituido a los sistemas telefónicos de audio tradicionales. (Por ejemplo, DSL ofrece tasas de transferencia del orden de 54 Mbps.)

La tasa máxima disponible en cada sistema concreto dependerá del tipo de enlace de comunicaciones utilizado y de la tecnología empleada para su implementación. Esta tasa máxima suele identificarse, de una forma un tanto aproximada, con el **ancho de banda** del enlace de comunicaciones, aunque el término *ancho de banda* también tiene connotaciones de capacidad más que de tasa de transferencia. Es decir, cuando afirmamos que un enlace de comunicaciones tiene un gran ancho de banda (o que proporciona un servicio de **banda ancha**) significa que el enlace de comunicaciones tiene la capacidad de transferir bits con una alta velocidad, así como la capacidad de transportar grandes cantidades de información simultáneamente.

Cuestiones y ejercicios

1. Suponga que la máquina descrita en el Apéndice C utiliza E/S mapeada en memoria y que la dirección B5 es la ubicación dentro del puerto de impresora al que hay que enviar los datos que se quieren imprimir.

- **a.** Si el registro 7 contiene el código ASCII correspondiente a la letra A, ¿qué instrucción del lenguaje máquina habría que emplear para que esa letra se imprimiera en la impresora?
- **b.** Si la máquina ejecuta un millón de instrucciones por segundo, ¿cuántas veces puede enviarse este carácter a la impresora en un segundo?
- **c.** Si la impresora es capaz de imprimir cinco páginas tradicionales de texto por minuto, ¿sería capaz de imprimir todos los caracteres que se le enviaran, según la respuesta al apartado (b)?

2. Suponga que el disco duro de su computadora personal gira a 3000 revoluciones por minuto, que cada pista consta de 16 sectores y que cada sector contiene 1024 bytes. ¿Qué velocidad de comunicación aproximada se requiere entre la unidad de disco y la controladora de disco, si esta última debe recibir los bits de la unidad de disco a medida que estos son leídos mientras el disco gira?
3. Estime el tiempo que se tardaría en transferir una novela de 300 páginas codificada en Unicode con una tasa de transferencia de 54 Mbps.

2.6 Otras arquitecturas

Para ampliar la perspectiva de nuestro estudio, vamos a considerar algunas alternativas a la arquitectura de computadoras tradicional que hemos analizado hasta el momento.

Cauce segmentado

Los pulsos eléctricos viajan a través de un cable a una velocidad que no puede ser mayor que la de la luz. Puesto que la luz recorre aproximadamente 30 centímetros en un nanosegundo (una mil millonésima de segundo), se requieren al menos 2 nanosegundos para que el procesador extraiga una instrucción de una celda de memoria que esté a 30 centímetros de distancia (la solicitud de lectura debe enviarse a la memoria, lo que requiere al menos 1 nanosegundo y la instrucción debe ser devuelta al procesador, lo que requiere al menos otro nanosegundo). En consecuencia, para extraer y ejecutar una instrucción en dicho tipo de máquina se necesitan varios nanosegundos, lo que implica que el incrementar la velocidad de ejecución de una máquina termina transformándose en un problema de miniaturización.

Sin embargo, el incrementar la velocidad de ejecución no es la única manera de mejorar el rendimiento de una computadora. El objetivo real consiste en aumentar la **tasa de procesamiento** de la máquina, que hace referencia a la cantidad total de trabajo útil que la máquina puede realizar en un determinado periodo de tiempo.

Un ejemplo de cómo se puede incrementar la tasa de procesamiento de una computadora sin necesidad de aumentar la velocidad de ejecución sería la técnica denominada **segmentación de cauce** (*pipelining*), que es la técnica consistente en permitir que se solapen los distintos pasos que componen el ciclo de máquina. En particular, mientras una instrucción está ejecutándose, puede irse extrayendo la siguiente instrucción, lo que quiere decir que puede haber más de una instrucción en el "cauce de procesamiento" en cualquier momento dado, estando cada una de esas instrucciones en una etapa diferente de su procesamiento. Esto hace que la tasa total de procesamiento de la máquina se incremente, aunque el tiempo requerido para extraer y ejecutar cada instrucción individual siga siendo el mismo. (Por supuesto, cuando se llega a una instrucción JUMP, cualquier ganancia que se hubiera obtenido gracias a la extracción anticipada de las instrucciones no llega a materializarse,

porque las instrucciones contenidas en el "cauce de procesamiento" no son las que deben ejecutarse a continuación.)

Los diseños modernos de máquinas aprovechan el concepto de segmentación de cauce en formas que van más allá de lo que este ejemplo simple permite intuir. Las máquinas modernas son capaces a menudo de extraer varias instrucciones al mismo tiempo y de ejecutar más de una instrucción simultáneamente, siempre y cuando esas instrucciones no dependan unas de otras.

Máquinas multiprocesador

La segmentación de cauce puede considerarse como una especie de primer paso hacia el **procesamiento paralelo**, que es la técnica consiste en realizar varias actividades al mismo tiempo. Sin embargo, el verdadero procesamiento paralelo requiere más de una unidad de procesamiento, lo que da como resultado las computadoras conocidas con el nombre de máquina multiprocesador.

Algunas computadoras actuales se diseñan con esta idea en mente. Una posible estrategia consiste en conectar con la misma memoria principal varias unidades de procesamiento, cada una de las cuales se asemeja al procesador de una máquina monoprocesador. Con esta configuración, los procesadores pueden operar de manera independiente, sin por ello dejar de coordinar sus esfuerzos, dejándose mensajes unos a otros en las celdas de memoria comunes. Por ejemplo, cuando un procesador se enfrenta a una tarea de gran envergadura, puede almacenar un programa que implemente parte de dicha tarea en la memoria común y luego solicitar a otro procesador que lo ejecute. El resultado es una máquina en la que se ejecutan diferentes secuencias de instrucciones para distintos conjuntos de datos, lo que se denomina arquitectura **MIMD** (*Multiple-instruction stream, Multiple-data stream*), por oposición a la arquitectura más tradicional **SISD** (*Single-instruction stream, Single-data stream*), que opera con un único flujo de instrucciones y un único flujo de datos.

Procesadores multinúcleo

A medida que la tecnología proporciona formas de integrar cada vez más circuitería en un chip de silicio, la distinción física entre los componentes de una computadora se diluye. Por ejemplo, un mismo chip puede contener un procesador y una memoria principal. Este es un ejemplo de la técnica denominada "sistema en un chip" en la que el objetivo es proporcionar un aparato completo en un único dispositivo que pueda ser utilizado como herramienta abstracta en diseños de mayor nivel. En otros casos, se proporcionan múltiples copias del mismo circuito dentro de un único dispositivo. Esta táctica comenzó a utilizarse en forma de chips que contenían varias puertas independientes o múltiples biestables. El estado actual de la técnica permite integrar en un mismo chip más de un procesador y esta es la arquitectura subyacente de los dispositivos conocidos con el nombre de procesadores multinúcleo, que están compuestos por dos o más procesadores que residen en el mismo chip junto con una memoria caché compartida. (Los procesadores multinúcleo que contienen dos unidades de procesamiento se suelen denominar procesadores de doble núcleo.) Dichos dispositivos simplifican la construcción de sistemas MIMD y se utilizan ya ampliamente en las computadora domésticas.

Una variante de la arquitectura multiprocesador consiste en enlazar los procesadores entre sí, para que puedan ejecutar la misma secuencia de instrucciones al unísono, cada uno con su propio conjunto de datos. Esto conduce a la arquitectura **SIMD** (*Single-instruction stream, Multiple-data stream*). Dichas máquinas son útiles para aquellas aplicaciones en las que se tenga que llevar a cabo la misma tarea con cada conjunto de elementos similares dentro de un bloque de datos de gran tamaño.

Otra técnica de procesamiento paralelo consiste en construir computadoras de gran tamaño como conglomerados de máquinas más pequeñas, cada una con su propia memoria y su propio procesador. Con este tipo de arquitectura, cada una de las máquinas de pequeño tamaño está acoplada a sus vecinas, de modo que las tareas asignadas al sistema completo pueden repartirse entre las máquinas individuales. Así, si una de las tareas asignadas a una de las máquinas internas puede descomponerse en subtareas independientes, dicha máquina puede pedir a sus vecinas que realicen de manera concurrente dichas subtareas. La tarea original puede así completarse en mucho menos tiempo que el que se requeriría utilizando una máquina con un único procesador.

Cuestiones y ejercicios

1. Consulte de nuevo la Cuestión 3 de la Sección 2.3. Si la máquina utilizara la técnica de cauce segmentado explicada en el texto, ¿qué habría en el "cauce de procesamiento" cuando se ejecutara la instrucción situada en la dirección AA? ¿En qué condiciones la segmentación de cauce no proporcionaría ninguna ventaja en dicho punto del programa?

2. ¿Qué conflictos habrá que resolver al ejecutar el programa de la Cuestión 4 de la Sección 2.3 en una máquina con arquitectura de cauce segmentado?

3. Suponga que tenemos dos unidades de procesamiento "centrales" conectadas a la misma memoria y ejecutando programas diferentes. Suponga también que uno de esos procesadores necesita sumar una unidad al contenido de una celda de memoria, aproximadamente al mismo tiempo que el otro procesador necesita restar una unidad del contenido de la misma celda de memoria (el efecto neto debería ser que la celda terminara teniendo el mismo valor con el que empezó).

   a. Describa una secuencia en la que estas actividades harían que la celda terminara teniendo un valor una unidad inferior a su valor inicial.
   b. Describa una secuencia en la que estas actividades harían que la celda terminara teniendo un valor una unidad superior a su valor inicial.

Problemas de repaso

(Los problemas con asterisco están asociados con las secciones opcionales.)

1. a. ¿Qué encontrará en el interior de un procesador, registros de uso general o celdas de la memoria principal?
   b. ¿Qué es un microprocesador?

2. Responda a las siguientes preguntas en términos del lenguaje máquina descrito en el Apéndice C.
   a. Escriba la instrucción 2103 (hexadecimal) como una cadena de 16 bits.
   b. Escriba el código de operación de la instrucción A324 (hexadecimal) como una cadena de 4 bits.
   c. Escriba el campo de operando de la instrucción A234 (hexadecimal) como una cadena de 12 bits.

3. Suponga que hay un bloque de datos almacenado en las celdas de memoria de la máquina descrita en el Apéndice C, desde la dirección BB a la C4, ambas incluidas. ¿Cuántas celdas de memoria hay en este bloque? Indique sus direcciones.

4. ¿Cuáles son los distintos tipos de registros? ¿Cuáles son los usos de los registros del procesador y de los registros de uso general?

5. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 05, contienen los siguientes patrones de bits:

Dirección	Contenido
00	22
01	11
02	32
03	02
04	C0
05	00

   Suponiendo que inicialmente el contador de programa contiene el valor 00, indique el contenido del contador de programa, del registro de instrucciones, y de la celda de memoria situada en la dirección 02 al final de la fase de captación de cada ciclo de máquina hasta que la máquina se detiene.

6. Suponga que hay tres valores: a, b y c almacenados en la memoria de una máquina. Describa la secuencia de sucesos (carga de registros desde la memoria, almacenamiento de valores en la memoria, etc.) que permite realizar el cálculo de $a - b + c$. ¿Y en el caso de $(2a) + (2c)$?

7. Las siguientes instrucciones están escritas en el lenguaje máquina descrito en el Apéndice C. Tradúzcalas a castellano.
   a. 7123 b. 40E1 c. A304
   d. B100 e. 2BCD

8. Suponga que disponemos de un lenguaje máquina diseñado con un campo de código de operación de 6 bits. ¿Cuántos tipos diferentes de instrucciones puede tener este lenguaje? ¿Y si se incrementa la longitud del campo de código de operación a 8 bits?

9. Traduzca las siguientes instrucciones expresadas en lenguaje normal al lenguaje máquina descrito en el Apéndice C.
   a. Cargar (LOAD) el registro 6 con el valor hexadecimal 77.
   b. Cargar (LOAD) el registro 7 con el contenido de la celda de memoria 77.
   c. Saltar (JUMP) a la instrucción situada en la posición de memoria 24 si el contenido del registro 0 es igual al valor contenido en el registro A.
   d. Rotar (ROTATE) el registro 4 tres bits hacia la derecha.
   e. Combinar mediante AND los contenidos de los registros E y 2, y almacenar el resultado en el registro 1.

10. La instrucción “Si 1 es igual a 1, entonces saltar al paso 10”, ¿es un salto condicional o incondicional? Explique su respuesta.

11. Clasifique cada una de las siguientes instrucciones (en el lenguaje máquina descrito en el Apéndice C) en términos de si su ejecución modifica el contenido de la celda de memoria situada en la posición 3C, de si extrae el contenido de la celda de memoria situada en la posición 3C o es indepen-

diente del contenido de la celda de memoria cuya dirección es 3C.

a. 353C b. 253C c. 153C
d. 3C3C e. 403C

12. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 03, contienen los siguientes patrones de bits:

Dirección	Contenido
00	26
01	55
02	C0
03	00

a. Exprese la primera instrucción en lenguaje coloquial.
b. Si la máquina arranca con su contador de programa conteniendo 00, ¿qué patrón de bits contendrá el registro 6 cuando la máquina se detenga?

13. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 02, contienen los siguientes patrones de bits:

Dirección	Contenido
00	12
01	21
02	34

a. ¿Cuál sería la primera instrucción ejecutada si hemos arrancado la máquina teniendo el valor 00 en el contador de programa?
b. ¿Cuál sería la primera instrucción ejecutada si hubiéramos arrancado la máquina teniendo el valor 01 en el contador de programa?

14. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 05, contienen los patrones de bits de la tabla que sigue.

A la hora de responder a las preguntas siguientes, suponga que la máquina inicia sus operaciones teniendo el valor 00 en su contador de programa.

a. Escriba las instrucciones que van a ejecutarse en lenguaje coloquial.

Dirección	Contenido
00	12
01	02
02	32
03	42
04	C0
05	00

b. ¿Qué patrón de bits contendrá la celda de memoria situada en la dirección 42 cuando la máquina se detenga?
c. ¿Qué patrón de bits contendrá el contador de programa cuando la máquina se detenga?

15. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 09, contienen los siguientes patrones de bits:

Dirección	Contenido
00	1C
01	03
02	2B
03	03
04	5A
05	BC
06	3A
07	00
08	C0
09	00

Suponga que la máquina inicia sus operaciones conteniendo el valor 00 en el contador de programa.

a. ¿Cuál será el contenido de la celda de memoria situada en la dirección 00 cuando la máquina se detenga?
b. ¿Qué patrón de bits contendrá el contador de programa cuando la máquina se detenga?

16. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 07, contienen los patrones de bits de la siguiente tabla.

a. Indique las direcciones de las celdas de memoria que almacenan el programa que se ejecutará si la máquina comienza a operar teniendo el valor 00 en su contador de programa.

Dirección	Contenido
00	2B
01	07
02	3B
03	06
04	C0
05	00
06	00
07	23

b. Enumere las direcciones de las celdas de memoria que se utilizan para almacenar los datos.

17. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 0D, contienen los siguientes patrones de bits:

Dirección	Contenido
00	20
01	04
02	21
03	01
04	40
05	12
06	51
07	12
08	B1
09	0C
0A	B0
0B	06
0C	C0
0D	00

Suponga que al arrancar la máquina su contador de programa contiene el valor 00.

a. ¿Qué patrón de bits contendrá el registro 0 cuando la máquina se detenga?

b. ¿Qué patrón de bits contendrá el registro 1 cuando la máquina se detenga?

c. ¿Qué patrón de bits contendrá el contador de programa cuando la máquina se detenga?

18. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de F0 a FD, contienen los patrones de bits (hexadecimal) indicados en la siguiente tabla.

Dirección	Contenido
F0	20
F1	00
F2	22
F3	02
F4	23
F5	04
F6	B3
F7	FC
F8	50
F9	02
FA	B0
FB	F6
FC	C0
FD	00

Si la máquina inicia sus operaciones cuando su contador de programa contiene el valor F0, ¿cuál será el valor almacenado en el registro 0 cuando finalmente la máquina ejecute la instrucción de detención situada en la posición FC?

19. ¿Cuáles son los pasos que deben seguirse en una computadora cuando se mueve el contenido de una celda de memoria a otra celda de memoria?

20. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 20 a 28, contienen los siguientes patrones de bits:

Dirección	Contenido
20	12
21	20
22	32
23	30
24	B0
25	21
26	24
27	C0
28	00

Suponga que al arrancar la máquina su contador de programa contiene el valor 20.

a. ¿Qué patrones de bits contendrán los registros 0, 1 y 2 cuando la máquina se detenga?

b. ¿Qué patrón de bits contendrá la celda de memoria situada en la dirección 30 cuando la máquina se detenga?

c. ¿Qué patrón de bits contendrá la celda de memoria situada en la dirección B0 cuando la máquina se detenga?

21. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de AF a B1, contienen los siguientes patrones de bits:

Dirección	Contenido
AF	B0
B0	B0
B1	AF

¿Qué ocurrirá si iniciáramos la máquina cuando su contador de programa contiene el valor AF?

22. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 05, contienen los siguientes patrones de bits: (hexadecimal):

Dirección	Contenido
00	25
01	B0
02	35
03	04
04	C0
05	00

Si en el momento de iniciarse la máquina el contador de programa contiene el valor 00, ¿cuándo se detendrá la máquina?

23. Para cada uno de los casos siguientes, escriba un programa corto en el lenguaje máquina descrito en el Apéndice C para llevar a cabo las actividades solicitadas. Suponga que cada uno de esos programas se almacena en memoria a partir de la dirección 00.

a. Mover el valor que se encuentra en la posición de memoria D8 a la posición de memoria B3.
b. Intercambiar los valores almacenados en las posiciones de memoria D8 y B3.
c. Si el valor almacenado en la posición de memoria 44 es 00, entonces escribir el valor 01 en la posición de memoria 46; en caso contrario, escribir el valor FF en la posición de memoria 46.

24. Un juego que solía ser muy popular entre los aficionados a las computadoras era las guerras de núcleo, una variante del típico juego de la batalla naval. (El término ***núcleo*** tiene su origen en una antigua tecnología de memoria en la que los 0s y 1s se representaban como campos magnéticos en pequeños anillos de material magnético. A esos anillos se les llamaba núcleos.) El juego tiene lugar entre dos programas oponentes, cada uno de ellos almacenado en una ubicación diferente dentro de la memoria de una misma computadora. Lo que la computadora hace es alternar entre los dos programas, ejecutando una instrucción de uno de ellos seguida de una instrucción del otro. El objetivo de cada programa es hacer que el otro funcione mal escribiendo sobre él datos sin sentido; sin embargo, ninguno de los programas conoce la ubicación en la que está el otro.

a. Escriba un programa en el lenguaje máquina descrito en el Apéndice C que trate de encontrar una solución con carácter defensivo intentando ser lo más pequeño posible.
b. Escriba un programa en el lenguaje máquina descrito en el Apéndice C que trate de evitar los ataques del programa oponente, moviéndose a diferentes ubicaciones. Para ser más precisos, escriba un programa que, comenzando en la posición 00, se copie a sí mismo en la posición 70 y luego salte a la posición 70.
c. Amplíe el programa escrito en el apartado (b) para continuar reubicándose en nuevas posiciones de la memoria. En particular, haga que su programa se mueva a la dirección 70, luego a la dirección E0 (= 70 + 70), luego a la 60 (= 70 + 70 + 70), etc.

25. ¿Qué es la unidad central de procesamiento (CPU)? ¿Cuáles son los diferentes componentes de una CPU? Indique cuál es la función de cada uno de los componentes. Lógicamente, ¿la memoria caché está colocada entre los registros y qué otro tipo de memoria? ¿Cuál es la ventaja de utilizar la memoria caché?

26. Suponga que las celdas de memoria de la máquina descrita en el Apéndice C, cuyas direcciones van de 00 a 05, contienen los siguientes patrones de bits (hexadecimal):

Dirección	Contenido
00	20
01	C0
02	30
03	04
04	00
05	00

¿Qué sucede si la máquina comienza a operar conteniendo el valor 00 en el contador de programa?

27. ¿Cuál es la diferencia entre los dispositivos de almacenamiento masivo, la memoria principal y los registros de uso general? En las primeras computadoras, los pasos ejecutados por cada dispositivo estaban incorporados a la unidad de control como parte de la máquina. ¿Cómo eliminó esta limitación el concepto de programa almacenado?

28. Suponga que el siguiente programa, escrito en el lenguaje máquina descrito en el Apéndice C, está almacenado en la memoria principal a partir de la dirección 30 (hexadecimal). ¿Qué tarea realizará el programa cuando se ejecute?

```
2003
2101
2200
2310
1400
3410
5221
5331
3239
333B
B248
B038
C000
```

29. ¿Cuáles son las distintas partes de las instrucciones del lenguaje máquina? Explique los usos de la operación STORE y del registro contador de programa.

***30.** Defina un caso en el que el uso del Acceso directo a memoria (DMA) puede ser perjudicial. ¿Qué es el cuello de botella de von Neumann?

***31.** ¿Cuáles son las características de salida de una DSL? ¿Qué unidades de medida se utilizan para calcular las velocidades de comunicación? ¿Qué significa una velocidad de comunicación de 1 Kbps?

***32.** Suponga que los registros 4 y 5 de la máquina descrita en el Apéndice C contienen los patrones de bits 3A y C8, respectivamente. ¿Qué patrón de bits quedará almacenado en el registro 0 después de ejecutar cada una de las siguientes instrucciones:

a. 5045 b. 6045 c. 7045
d. 8045 e. 9045

***33.** Utilizando el lenguaje máquina descrito en el Apéndice C, escriba una serie de programas para llevar a cabo cada una de las siguientes tareas:

a. Copiar en la posición de memoria AA el patrón de bits almacenado en la posición de memoria 44.
b. Cambiar los 4 bits menos significativos de la celda de memoria situada en la posición 34 a 0s, dejando intactos el resto de los bits.
c. Tomar los 4 bits menos significativos de la celda de memoria situada en la posición A5 y copiarlos en los 4 bits menos significativos de la posición A6, dejando intactos los restantes bits de la posición A6.
d. Tomar los 4 bits menos significativos de la posición de memoria A5 y copiarlos en los 4 bits más significativos de la posición de memoria A5. (Por tanto, los primeros 4 bits de la posición serán iguales que los últimos 4 bits.)

***34.** Realice las operaciones indicadas:

```
a.      111001      b.      000101
    AND 101001          AND 101010
    ----------          ----------

c.      001110      d.      111011
    AND 010101          AND 110111
    ----------          ----------

e.      111001      f.      010100
    OR  101001          OR  101010
    ----------          ----------
```

```
g.       000100      h.       101010
     OR  010101           OR  110101

i.       111001      j.       000111
    XOR  101001          XOR  101010

k.       010000      l.       111111
    XOR  010101          XOR  110101
```

***35.** Identifique tanto la máscara como la operación lógicas necesarias para llevar a cabo cada una de las siguientes tareas:
   a. Poner una serie de 1s en los 4 bits más significativos de un patrón de 8 bits dejando intactos los bits restantes.
   b. Complementar el bit más significativo de un patrón de 8 bits sin modificar los restantes bits.
   c. Complementar un patrón de 8 bits.
   d. Poner un 0 en el bit menos significativo de un patrón de 8 bits, sin modificar los restantes bits.
   e. Poner 1s en todos los bits de un patrón de 8 bits, excepto en el más significativo, sin modificar dicho bit más significativo.

***36.** ¿Con qué nombre se conoce a la capacidad de una controladora para acceder a la memoria principal? ¿Cómo mejora esta capacidad el rendimiento de una computadora?

***37.** Diseñe un sistema y resuma los pasos necesarios para sumar dos valores almacenados en la memoria principal. Diseñe otro sistema y resuma los pasos necesarios para dividir dos valores almacenados en la memoria principal, teniendo en cuenta la división por cero.

***38.** ¿Cuál sería el resultado de realizar un desplazamiento circular hacia la izquierda de 4 bits en los siguientes patrones de bits?
   a. 10101 b. 11110000 c. 001
   d. 101000 e. 00001

***39.** ¿Cuál sería el resultado de realizar un desplazamiento circular hacia la derecha de 2 bits en los siguientes bytes representados en notación hexadecimal (exprese su respuesta en notación hexadecimal)?
   a. 3F b. 0D
   c. FF d. 77

***40.** a. ¿Qué única instrucción del lenguaje máquina del Apéndice C podría utilizarse para realizar un desplazamiento circular hacia la derecha de 5 bits del registro B?
   b. ¿Qué única instrucción del lenguaje máquina del Apéndice C podría utilizarse para realizar un desplazamiento circular hacia la izquierda de 2 bits del registro B?

***41.** Escriba un programa en el lenguaje máquina descrito en el Apéndice C que invierta el contenido de la celda de memoria situada en la dirección 8C. (Es decir, el patrón de bits final contenido en la dirección 8C debe concordar, al leerlo de izquierda a derecha, con el patrón original leído de derecha a izquierda.)

***42.** Escriba un programa en el lenguaje máquina descrito en el Apéndice C que reste el valor almacenado en A1 del valor almacenado en la dirección A2 y almacene el resultado en la dirección A0. Suponga que los valores están codificados en notación de complemento a dos.

***43.** El vídeo de alta definición puede ser suministrado a una velocidad de 30 imágenes por segundo, teniendo cada imagen una resolución de 1920×1080 píxeles y utilizándose 24 bits por píxel. ¿Puede enviarse un flujo de vídeo de este formato y no comprimido a través de un puerto serie USB 1.1? ¿Y a través de un puerto serie USB 2.0? ¿Y a través de un puerto serie USB 3.0? (Nota: las velocidades máximas de los puertos serie USB 1.1, USB 2.0 y USB 3.0 son 12 Mbps, 480 Mbps y 5 Gbps, respectivamente.)

***44.** Suponga que una persona está escribiendo en un teclado cuarenta palabras por minuto (suponemos que una palabra está formada por 5 caracteres). Si una máquina ejecuta 500 instrucciones cada microsegundo (una millonésima de segundo), ¿cuántas instrucciones ejecuta la máquina durante el tiempo que transcurre entre la escritura de dos caracteres consecutivos?

***45.** ¿Cuántos bits por segundo debe transmitir un teclado para no quedarse rezagado con respecto a una persona que está escribiendo cuarenta palabras por minuto? Suponga que cada carácter se codifica en ASCII y que cada palabra consta de seis caracteres.

***46.** Suponga que la máquina descrita en el Apéndice C se comunica con una impresora utilizando la técnica de la E/S mapeada en memoria. Suponga también que se emplea la dirección FF para enviar caracteres a la impresora y la dirección FE para recibir información acerca del estado de la impresora. En particular, suponga que el bit menos significativo de la dirección FE indica si la impresora está lista para recibir otro carácter (un 0 indica que "no está lista" y un 1 indicaría que "está lista"). Comenzando en la dirección 00, escriba una rutina en lenguaje máquina que espere a que la impresora esté lista para recibir otro carácter y a continuación le envíe el carácter representado por el patrón de bits contenido en el registro 5.

***47.** Escriba un programa en el lenguaje máquina descrito en el Apéndice C que coloque el valor 0 en todas las celdas de memoria cuyas direcciones van de A0 a C0 pero que sea lo suficientemente pequeño como para caber en las celdas de memoria que van de la dirección 00 a la 13 (hexadecimal).

***48.** Suponga que una máquina tiene 200 GB de espacio de almacenamiento en un disco duro y recibe datos a través de una conexión de banda ancha con una velocidad de 15 Mbps. Con esta velocidad, ¿cuánto se tardará en llenar el espacio de almacenamiento disponible?

***49.** Suponga que estamos utilizando un sistema vía satélite para recibir un flujo de datos en serie a 250 Kbps. Si una ráfaga de interferencia atmosférica dura 6,96 segundos, ¿cuántos bits de datos se verán afectados?

***50.** Suponga que nos dan 32 procesadores, cada uno de ellos capaz de calcular la suma de dos números multidígito en una millonésima de segundo. Describa cómo podríamos aplicar técnicas de procesamiento en paralelo para calcular la suma de 64 números en solo la seismillonésima parte de un segundo. ¿Cuánto tiempo necesitaría un único procesador para calcular esta misma suma?

***51.** Explique la diferencia entre una arquitectura MIMD y una arquitectura SISD.

***52.** Indique cómo la técnica de cauce segmentado aumentaría la capacidad de procesamiento.

***53.** Identifique el tipo de arquitectura que sería apropiada para:
a. una máquina que sea eficiente, rápida y poco cara de fabricar.
b. una máquina que sea capaz de hacer frente a la complejidades cada vez mayores del software actual.

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Suponga que un fabricante de computadoras desarrolla una nueva arquitectura de máquina. ¿Hasta qué punto habría que permitir que esa empresa sea la propietaria de dicha arquitectura? ¿Qué política sería la mejor para la sociedad?

2. En un cierto sentido, el año 1923 marcó el nacimiento de lo que ahora se conoce con el nombre de ***obsolescencia planificada.*** Ese fue el año en el que General Motors, dirigida por Alfred Sloan, introdujo el concepto de año de los modelos en la industria del automóvil. La idea era incrementar las ventas cambiando el estilo, sin necesidad de fabricar necesariamente un automóvil mejor. A Sloan se le atribuye la frase "Queremos hacer que te sientas insatisfecho con tu vehículo actual, para que compres uno nuevo". ¿Hasta qué punto se utiliza esta técnica de marketing hoy día dentro del sector de la computación?
3. A menudo pensamos en la manera en que la tecnología de la computación ha cambiado nuestra sociedad. Muchas personas afirman, sin embargo, que esta tecnología ha evitado a menudo que se produjeran cambios, al permitir sobrevivir a los antiguos sistemas y, en algunos casos, al hacerlos más fuertes. Por ejemplo, ¿habría sobrevivido el papel del gobierno central en nuestra sociedad actual sin la tecnología de computación? ¿Hasta qué punto estaría presente una autoridad centralizada actualmente si la tecnología de la computación no estuviera disponible? ¿Hasta qué punto estaríamos mejor o peor que ahora si no existiera la tecnología de la computación?
4. ¿Cree que es ético que una persona tome la aptitud de que no necesita saber nada acerca de los detalles internos de una máquina, porque ya habrá otras personas que la construyan, la mantengan y arreglen los problemas que puedan producirse? ¿Depende su respuesta de que la máquina sea una computadora, un automóvil, una central nuclear o una tostadora?
5. Suponga que un fabricante fabrica un chip de computadora y descubre posteriormente un fallo en su diseño. Suponga también que el fabricante corrige el fallo en los lotes de producción sucesivos, pero decide mantener en secreto el fallo original y no reclama los chips ya vendidos, utilizando el razonamiento de que ninguno de los chips que se está utilizando está siendo empleado en una aplicación que el fallo pueda tener consecuencias. ¿Hay alguien que salga perjudicado por la decisión de ese fabricante? ¿Está justificada la decisión del fabricante si nadie resulta dañado y si esa decisión impide que el fabricante pierda dinero y, posiblemente, despida a empleados?
6. ¿Cree que los avances de la tecnología proporcionan remedios para las dolencias cardiacas o que inducen a un estilo de vida sedentario que contribuye a que aumenten dicho tipo de dolencias?
7. Es fácil imaginar desastres financieros o de navegación que pueden producirse como resultado de errores aritméticos debidos a problemas de desbordamientos y truncamientos. ¿Qué consecuencias podrían derivarse de los errores producidos en sistemas de almacenamiento de imágenes debidos a la pérdida de detalles de esas imágenes (tal vez en campos tales como el reconocimiento o el diagnóstico médico)?
8. ARM Holdings es una empresa pequeña que diseña procesadores para una amplia variedad de dispositivos electrónicos de consumo. No fabrica ninguno de esos procesadores; en lugar de ello lo que hace es licenciar los diseños a fabricantes de semiconductores (tales como Qualcomm, Samsung y Texas

Instruments) que pagan un royalty por cada unidad fabricada. Este modelo de negocio distribuye el alto coste de investigación y desarrollo de procesadores para computadora en todo el mercado de la electrónica de consumo. Hoy día, más del 95 por ciento de todos los teléfonos celulares (no solo los teléfonos inteligentes), más del 40 por ciento de todas las cámaras digitales y el 25 por ciento de las televisiones digitales utilizan un procesador ARM. Además, los procesadores ARM pueden encontrarse en mini-portátiles, reproductores MP3, controladoras de juegos, libros electrónicos, sistemas de navegación y muchos otros tipos de dispositivos. Teniendo esto en cuenta, ¿considera que esta empresa es un monopolio? ¿Por qué? Puesto que los dispositivos de consumo desempeñan un papel cada vez más importante en la sociedad actual, ¿es buena esta dependencia con respecto a esta empresa poco conocida o debería ser motivo de preocupación?

Lecturas adicionales

Carpinelli, J. D. *Computer Systems Organization and Architecture*. Boston, MA: Addison-Wesley, 2001.

Comer, D. E. *Essentials of Computer Architecture*. Upper Saddle River, NJ: Prentice- Hall, 2005.

Dandamudi, S P. *Guide to RISC Processors for Programmers and Engineers*. Nueva York: primavera, 2005.

Furber, S. *ARM System-on-Chip Architecture*, 2ª ed. Boston, MA: Addison-Wesley, 2000.

Hamacher, V. C., Z. G. Vranesic y S. G. Zaky. *Computer Organization,* 5ª ed. Nueva York: McGraw-Hill, 2002.

Knuth, D. E. *The Art of Computer Programming,* Vol. 1, 3ª ed. Boston, MA: Addison-Wesley, 1998.

Murdocca, M. J. y V. P. Heuring. *Computer Architecture and Organization: An Integrated Approach.* Nueva York: Wiley, 2007.

Stallings, W. *Computer Organization and Architecture,* 7ª ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Tanenbaum, A. S. *Structured Computer Organization*, 5ª ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

CAPÍTULO

3

Sistemas operativos

En este capítulo vamos a estudiar los sistemas operativos, que son paquetes software que coordinan las actividades internas de una computadora, además de controlar su comunicación con el mundo exterior. Un sistema operativo de una computadora es el que hace de la computadora una herramienta útil. Nuestro objetivo es comprender qué es lo que hacen los sistemas operativos y cómo lo hacen. Estos conocimientos son fundamentales para que el usuario pueda entender cómo funciona la computadora.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

Un **sistema operativo** es el software que controla el conjunto de operaciones de una computadora. Proporciona el mecanismo por el cual un usuario puede almacenar y extraer archivos, proporciona la interfaz mediante la que el usuario puede solicitar la ejecución de programas y proporciona también el entorno necesario para ejecutar los programas solicitados.

Quizá el ejemplo más conocido de sistema operativo sea Windows, que Microsoft proporciona en numerosas versiones y que se utiliza ampliamente en el mercado de los PC. Otro ejemplo bastante extendido es UNIX, muy empleado tanto en grandes sistemas de computación como en los PC. De hecho, UNIX forma el núcleo fundamental de otros dos sistemas operativos muy populares: Mac OS, que es el sistema operativo de Apple para su gama de computadoras Mac y Solaris, el cual fue desarrollado por Sun Microsystems (ahora propiedad de Oracle). Otro ejemplo de sistema operativo que puede encontrarse en máquinas tanto de pequeño como de gran tamaño es Linux, que fue originalmente desarrollado con carácter no comercial por varios entusiastas de las computadoras y ahora está disponible a través de muchos proveedores comerciales, incluyendo IBM.

Para los usuarios ocasionales de computadoras, las diferencias entre los sistemas operativos son más bien de carácter cosmético. Para los profesionales de la computación, sin embargo, los diferentes sistemas operativos pueden incluir variaciones enormemente importantes en las herramientas con las que trabajan o en la filosofía que deben seguir a la hora de diseminar y mantener su trabajo. De todos modos, en lo que respecta al núcleo fundamental de los sistemas operativos, todos los sistemas operativos más conocidos intentan resolver los mismos tipos de problemas con que los expertos en el campo de la computación se han enfrentado durante más de medio siglo.

3.1 Historia de los sistemas operativos

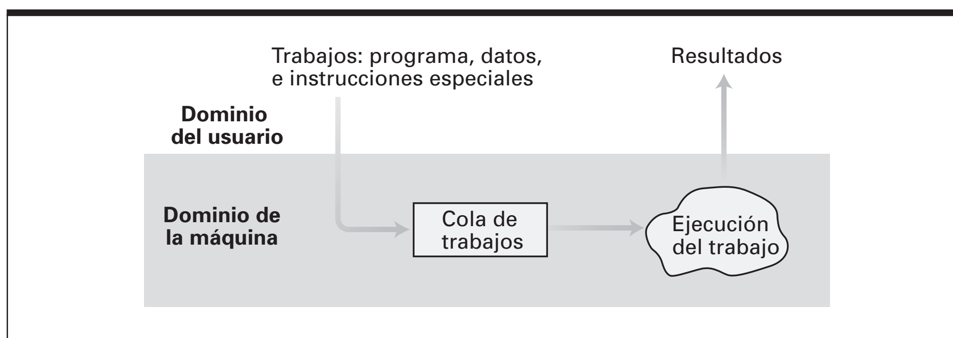
Los sistemas operativos actuales son paquetes software de gran tamaño y complejidad que han ido creciendo a partir de orígenes muy humildes. Las computadoras de las décadas de 1940 y 1950 no eran muy flexibles, ni tampoco eficientes, las máquinas podían ocupar una habitación completa. La ejecución de programas requería una tediosa preparación de los equipos, en el sentido de que había que montar cintas magnéticas, colocar tarjetas perforadas en los lectores de tarjetas, configurar una serie de conmutadores, etc. La ejecución de cada programa, denominada **trabajo**, se gestionaba como una actividad independiente: se preparaba la máquina para ejecutar el programa, se ejecutaba ese programa y luego había que extraer todas las cintas, tarjetas perforadas, etc. antes de que pudiera iniciarse la ejecución del siguiente programa. Cuando varios usuarios necesitaban compartir una máquina se suministraban hojas de petición para que esos usuarios pudieran reservar el uso de la máquina durante ciertos periodos de tiempo. Durante el periodo de tiempo asignado a un usuario, la máquina estaba totalmente bajo el control de dicho usuario. La sesión comenzaba normalmente con la preparación del programa, seguida de cortos periodos de ejecución de ese programa. A menudo, el usuario tenía que completar su tarea a toda prisa, tratando de terminar algún aspecto más de su proyecto ("Solo tardaré un minuto") mientras que el siguiente usuario se iba impacientando cada vez más mientras preparaba su propio programa.

En este tipo de entorno, los sistemas operativos vieron la luz como método para simplificar la preparación de los programas y para acelerar la transición entre un trabajo de programación y otro. Uno de los primeros desarrollos fue la separación de los usuarios y los equipos, que eliminó la necesidad de realizar una transición física entre programas con personas entrando y saliendo de la sala de computadoras. Con este fin, se contrató a un operador de computadora, para controlar el funcionamiento de la máquina. Si alguien quería ejecutar un programa, se le exigía que se lo enviara al operador junto con los datos necesarios y las instrucciones especiales relativas a los requisitos del programa, y que volviera después para recoger los resultados. El operador, por su parte, cargaba estos materiales en el almacenamiento masivo de la máquina, donde un programa denominado sistema operativo podía ir leyendo y ejecutando un programa cada vez. Este fue el comienzo de la técnica denominada **procesamiento por lotes**, la ejecución de trabajos recopilándolos en un único lote y luego ejecutándolos sin interacción adicional por parte del usuario.

En los sistemas de procesamiento por lotes, los trabajos que residen en los dispositivos de almacenamiento masivo esperan para ser ejecutados en una **cola de trabajos** (Figura 3.1). Una **cola** es una estructura de almacenamiento en la que los objetos (en este caso trabajos) están ordenados de acuerdo con la filosofía **primero en entrar, primero en salir** (FIFO, *first-in, first-out*). Es decir, los objetos se extraen de la cola en el mismo orden en que llegaron. En la realidad, la mayoría de las colas de trabajo no respetan de forma rigurosa la estructura FIFO, ya que la mayor parte de los sistemas operativos permiten asignar prioridades a los trabajos. Como resultado, un trabajo que esté a la espera en la cola de trabajos puede verse adelantado por otro trabajo de mayor prioridad.

En los primeros sistemas de procesamiento por lotes, cada trabajo iba acompañado de un conjunto de instrucciones que explicaban los pasos necesarios para preparar la máquina para ese trabajo concreto. Estas instrucciones se codificaban utilizando un sistema conocido como lenguaje de control de trabajos (JCL, ***Job Control Language***) y se almacenaba junto con el trabajo en la cola de trabajos. Cuando se seleccionaba ese trabajo para su ejecución, el sistema operativo imprimía estas instrucciones en una impresora para que el operador de la computadora pudiera leerlas y llevarlas a la práctica. Este tipo de comunicación entre el sistema operativo y el operador de la computadora sigue estando presente hoy día, como atestiguan esos sistemas operativos para PC

Figura 3.1 Procesamiento por lotes.

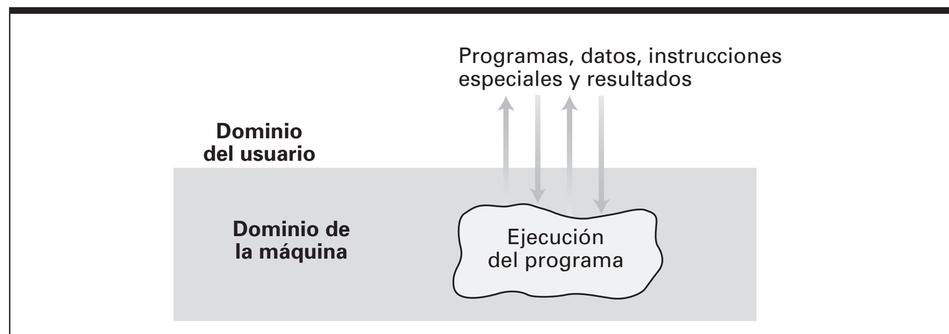


que informan de la existencia de errores tales como "unidad de disco no accesible" o "la impresora no responde". Una desventaja importante de utilizar al operador de la computadora como intermediario entre una computadora y los usuarios es que estos no tienen la posibilidad de interactuar con sus trabajos después de habérselos enviado al operador. Esta técnica es aceptable en algunas aplicaciones, como por ejemplo el procesamiento de nóminas, en las que los datos y todas las decisiones están prefijados de antemano. Sin embargo, no es aceptable cuando el usuario tiene que interactuar con un programa durante su ejecución, como sucede, por ejemplo, en los sistemas de reservas, en los que hay que informar de las reservas y cancelaciones a medida que se producen; o en los sistemas de procesamiento de textos en los que los documentos se desarrollan mediante un proceso de escritura y re-escritura dinámicas; o como en el caso de los juegos de computadora, en los que la interacción con la máquina es precisamente la característica fundamental del juego.

Para dar satisfacción a estas necesidades, se desarrollaron nuevos sistemas operativos que permitían que un programa en ejecución entablara un diálogo con el usuario a través de terminales remotos, una característica que se conoce con el nombre de **procesamiento interactivo** (Figura 3.2). (Un terminal era básicamente una máquina de escribir eléctrica, que permitía al usuario escribir las entradas necesarias y leer la respuesta que la computadora imprimía en un papel. Hoy día, los terminales han evolucionado hasta transformarse en dispositivos más sofisticados, denominados estaciones de trabajo, e incluso en computadoras PC completas que pueden funcionar como equipos autónomos si así se desea.)

Un requisito esencial para que el procesamiento interactivo pueda funcionar es que las acciones de la computadora sean lo suficientemente rápidas como para poder adaptarse a las necesidades del usuario, en lugar de obligar al usuario a adaptarse al horario de funcionamiento de la máquina. (La tarea de procesar las nóminas de una empresa puede programarse de manera que se adapte a la cantidad de tiempo requerida por la computadora, pero el uso de un procesador de textos resultaría frustrante si la máquina no respondiera inmediatamente a medida que se escriben los caracteres.) En un cierto sentido, la computadora está obligada a ejecutar las tareas con un límite de tiempo estricto, un proceso que se ha llegado a conocer con el nombre de **procesamiento en tiempo real**, en el que se dice que las acciones realizadas tienen lugar en tiempo real. Es decir, cuando afirmamos que una computadora realiza una tarea en tiempo real queremos decir

Figura 3.2 Procesamiento interactivo.



que la computadora lleva a cabo esa tarea de acuerdo con los límites de tiempo impuestos por su entorno (su entorno externo del mundo real).

Si los sistemas interactivos solo tuvieran que dar servicio a un usuario cada vez, el procesamiento en tiempo real no hubiera supuesto ningún problema. Pero las computadoras en las décadas de 1960 y 1970 eran muy caras, por lo que cada máquina debía dar servicio a más de un usuario. A su vez, era bastante común que varios usuarios trabajando en terminales remotos requirieran un servicio interactivo simultáneo de una misma máquina, por lo que las consideraciones de tiempo real presentaban un serio obstáculo. Si el sistema operativo insistiera en ejecutar únicamente un trabajo cada vez, solo un usuario podría recibir un servicio en tiempo real satisfactorio.

La solución a este problema fue diseñar sistemas operativos que proporcionaran un servicio simultáneo a múltiples usuarios. Una característica denominada **tiempo compartido**. Una forma de implementar la compartición de tiempo consiste en aplicar la técnica denominada **multiprogramación**, en la que el tiempo se divide en intervalos y dentro de cada intervalo solo se ejecuta un determinado trabajo. Al final de cada intervalo, el trabajo actual se pone temporalmente en espera, ejecutándose otro trabajo durante el siguiente intervalo. Conmutando rápidamente entre unos trabajos y otros de esta forma, se crea la ilusión de que hay varios trabajos que se están ejecutando de manera simultánea. Dependiendo de los tipos de trabajo que se ejecutaban, los primeros sistemas de tiempo compartido eran capaces de llevar a cabo un procesamiento en tiempo real aceptable para hasta 30 usuarios simultáneos. Hoy día, las técnicas de multiprogramación se emplean en sistemas tanto multiusuario como monousuario, aunque en este último caso, la técnica resultante se suele denominar **multitarea**. Es decir, la compartición de tiempo hace referencia al caso en el que tenemos múltiples usuarios compartiendo el acceso a una computadora común, mientras que la multitarea hace referencia a un usuario que está ejecutando varias tareas simultáneamente.

Con el desarrollo de los sistemas operativos multiusuario de tiempo compartido, una instalación de computadora típica se implementaba mediante una gran computadora central conectada a numerosas estaciones de trabajo. Desde estas estaciones de trabajo, los usuarios podían comunicarse directamente con la computadora desde fuera de la sala de computadoras, en lugar de mandar sus solicitudes a un operador. Los programas de uso más común estaban almacenados en los dispositivos de almacenamiento masivo de la máquina y se diseñaron sistemas operativos para ejecutar dichos programas según los fueran solicitando las estaciones de trabajo. A su vez, eso hizo que comenzara a desvanecerse el papel del operador de la computadora como intermediario entre esta y los usuarios.

Hoy día, los operadores de computadora han desaparecido casi completamente, especialmente en el campo de las computadoras personales en el que es el usuario de la computadora el que asume todas las responsabilidades del funcionamiento de la máquina. Incluso las más grandes instalaciones de computadora funcionan prácticamente de manera no atendida. De hecho, el puesto de operador de computadora ha dejado paso al de administrador de sistemas, que se encarga de gestionar el sistema de la computadora, contratando y supervisando la instalación de nuevos equipos y software; imponiendo las normas de funcionamiento vigentes, como por ejemplo abriendo nuevas cuentas de usuario y defi-

¿Qué contiene un teléfono inteligente?

A medida que ha ido aumentando la potencia de los teléfonos móviles, estos han comenzado a poder ofrecer servicios que van mucho más alla del simple procesamiento de llamadas de voz. Ahora, un **teléfono inteligente** típico puede emplearse para enviar mensajes de texto, navegar por la Web, acceder a mapas, ver contenido multimedia; en resumen, puede utilizarse para proporcionar muchos de los mismos servicios que un PC tradicional. Por esta razón, los teléfonos inteligentes requieren sistemas operativos completos, no solo para gestionar los limitados recursos del hardware del teléfono, sino también para poder ofrecer funcionalidades que den soporte al conjunto cada vez mayor de aplicaciones software para teléfonos inteligentes. La batalla por el predominio en el mercado de los sistemas operativos para teléfonos inteligentes promete ser muy cruenta y terminará ganándola, probablemente, el que pueda proporcionar las funcionalidades más imaginativas al mejor precio. Entre los competidores en el campo de los sistemas operativos para los teléfonos inteligentes podemos citar el iPhone OS de Apple, BlackBerry OS de Research In Motion, Windows Phone de Microsoft, Symbian OS de Nokia y Android de Google.

niendo los límites de almacenamiento masivo para los distintos usuarios; y coordinando esfuerzos para resolver los problemas que surjan en el sistema, en lugar de dedicarse a operar las máquinas con sus propias manos.

En resumen, los sistemas operativos han ido creciendo, pasando de ser simples programas que extraían y ejecutaban los programas de uno en uno a convertirse en sistemas complejos que coordinan la compartición de tiempo, mantienen los programas y los archivos de datos en los dispositivos de almacenamiento masivo y responden directamente a las solicitudes de los usuarios de la computadora.

Pero la evolución de los sistemas operativos continúa. El desarrollo de las máquinas multiprocesador ha conducido a la aparición de sistemas operativos que proporcionan capacidades de compartición de tiempo/multitarea, asignando diferentes tareas a diferentes procesadores, además de compartiendo el tiempo de cada uno de esos procesadores. Estos sistemas operativos deben enfrentarse a problemas tales como el **equilibrado de carga** (asignación dinámica de tareas a los diversos procesadores con el fin de utilizar todos ellos de manera eficiente), así como el **escalado** (descomposición de las tareas en una serie de subtareas compatible con el número de procesadores disponibles).

Además, la aparición de redes de computadoras en las que numerosas máquinas se conectan a larga distancia ha llevado a la creación de sistemas software para la coordinación de las actividades de la red. Desde el punto de vista de las redes (que estudiaremos en el Capítulo 4) constituye una extensión del tema de los sistemas operativos, siendo el objetivo principal el de gestionar los recursos para múltiples usuarios y múltiples máquinas, en lugar de trabajar con una única computadora aislada.

Otra de las líneas de investigación en el campo de los sistemas operativos se centra en los dispositivos dedicados a tareas específicas como los dispositivos médicos, la electrónica para automoción, los electrodomésticos, los teléfonos celulares y otras computadoras de mano. Los sistemas de computadoras que podemos encontrar en estos dispositivos se denominan **sistemas empotrados**. Los siste-

mas operativos empotrados suelen tener requisitos especiales, como que ayuden a ahorrar potencia de la batería, que se ajusten a límites de tiempo real muy estrictos o que operen de manera continua sin ninguna supervisión humana o con muy poca. Los mayores éxitos dentro de este campo son los de sistemas tales como VxWORKS, desarrollado por Wind River Systems y usado en los exploradores de la superficie de Marte llamados Spirit y Opportunity; Windows CE (también conocido como Pocket PC) desarrollado por Microsoft; y Palm OS desarrollado por PalmSource, Inc., especialmente para su uso en dispositivos de mano.

Cuestiones y ejercicios

1. Identifique ejemplos de colas. En cada caso, indique las posibles situaciones que hagan que se viole la estructura FIFO.
2. ¿Cuáles de las siguientes actividades requieren un procesamiento en tiempo real?
   - **a.** Impresión de etiquetas de correo.
   - **b.** Utilización de un juego de computadora.
   - **c.** Visualización de los números en la pantalla de un teléfono inteligente a medida que se marcan.
   - **d.** Ejecución de un programa que predice el estado de la economía durante el año próximo.
   - **e.** Reproducción de una grabación MP3.
3. ¿Cuál es la diferencia entre los sistemas empotrados y los PC?
4. ¿Cuál es la diferencia entre tiempo compartido y multitarea?

3.2 Arquitectura de un sistema operativo

Para entender la composición de un sistema operativo típico vamos a considerar primero la gama completa de software que vamos a encontrar en una computadora típica. Después nos concentraremos en el propio sistema operativo.

Un repaso al software

Vamos a tratar de dar un repaso al software que puede encontrarse en una computadora típica presentando un esquema de clasificación de ese software. La utilización de tales sistemas de clasificación presenta siempre el problema de que paquetes software similares terminan siendo asignados a distintas clases, de la misma forma que la división en zonas horarias obliga a que ciudades que están muy próximas entre sí deban regular sus relojes con una hora de diferencia, aún cuando en la práctica el amanecer y el ocaso tengan lugar prácticamente en el mismo momento en ambas poblaciones. Además, en el caso de la clasificación del software, la dinámica del propio mercado y la falta de una autoridad que defina los conceptos conduce a la aparición de terminología contradictoria. Por ejemplo, los usuarios de los sistemas operativos Windows de Microsoft disponen de grupos de programas denominados “Accesorios” y “Herramientas administrativas” que

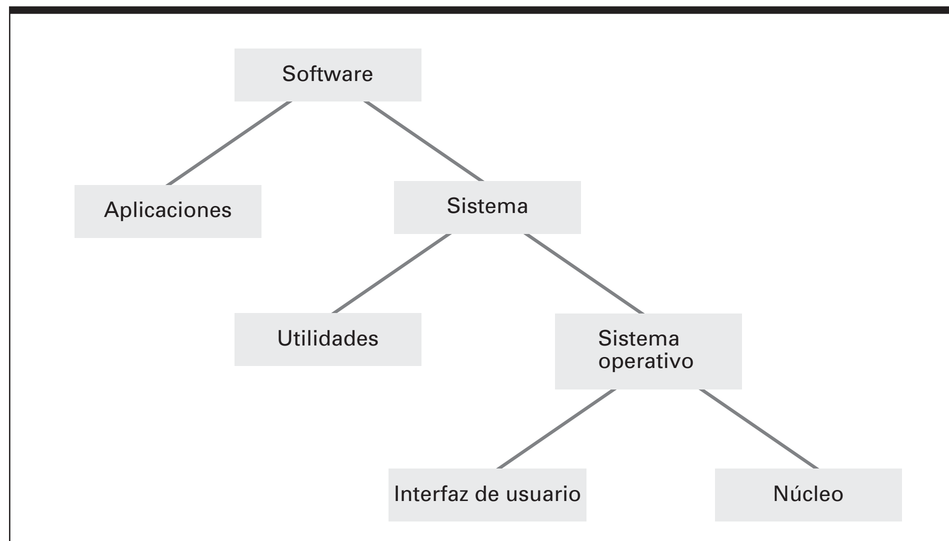
incluyen programas software de lo que nosotros denominamos las clases de aplicación y de utilidad. Por tanto, la taxonomía que vamos a presentar debe intepretarse como una manera de orientarse en un campo muy amplio y dinámico, más que considerarse la expresión de un hecho universalmente aceptado.

Comencemos dividiendo el software de una máquina en dos categorías muy amplias: **software de aplicación** y **software del sistema** (Figura 3.3). El software de aplicación está compuesto por todos los programas que realizan tareas relacionadas con la utilización concreta de la máquina. Una máquina empleada para mantener el inventario de una empresa de fabricación tendrá un software de aplicación distinto del que podremos encontrar en una máquina utilizada por un ingeniero eléctrico. Como ejemplos de software de aplicación podemos citar las hojas de cálculo, los sistemas de base de datos, los sistemas de autoedición, los sistemas de contabilidad, el software para el desarrollo de programas y los juegos.

A diferencia del software de aplicación, el software del sistema realiza las tareas que son comunes en general a todas las computadoras. En un cierto sentido, el software del sistema proporciona la infraestructura que necesita el software de aplicación, de forma bastante similar a como la infraestructura de un país (gobierno, carreteras, empresas de servicio público, instituciones financieras, etc.) proporciona la base de la que los ciudadanos dependen para llevar su propio estilo de vida individual.

Dentro de la clase software del sistema hay dos categorías: una es el propio sistema operativo y la otra está compuesta por unidades de software que se conocen colectivamente como **software de utilidad**. La mayor parte del software de utilidad de una instalación está compuesto por programas que se emplean para realizar actividades fundamentales para esa computadora, pero que no se incluyen en el sistema operativo. En cierto sentido, el software de utilidad está compuesto por unidades software que amplían (o personalizan) las capacidades del sistema operativo. Por ejemplo, la capacidad de forma-

Figura 3.3 Clasificación del software.



tear un disco magnético o de copiar un archivo desde un disco magnético a un CD no suele implementarse dentro del sistema operativo, sino que se proporciona por medio de un programa de utilidad. Otros ejemplos de software de utilidad incluyen el software necesario para comprimir y descomprimir datos, el software para reproducir presentaciones multimedia o el software para gestionar las comunicaciones por red.

La implementación de ciertas actividades como software de utilidad permite personalizar el software del sistema para adecuarlo a las necesidades de una instalación concreta con mayor facilidad que si se hubiera incluido esa funcionalidad en el sistema operativo. De hecho, es habitual encontrarse con empresas o personas que han modificado o ampliado el software de utilidad proporcionado originalmente con el sistema operativo de la máquina.

Lamentablemente, la distinción entre software de aplicación y software del sistema puede resultar muy vaga. Desde nuestro punto de vista, la diferencia estriba en si el paquete forma parte de la "infraestructura software" de la computadora. Por tanto, una nueva aplicación puede evolucionar hasta terminar siendo un software de utilidad si se convierte en una herramienta fundamental. Cuando todavía era un proyecto de investigación, el software para comunicarse a través de Internet se consideraba software de aplicación; hoy día, dichas herramientas son fundamentales a la hora de utilizar la mayoría de los PC y, por tanto, se clasificaría como software de utilidad.

La distinción entre el software de utilidad y el sistema operativo es igualmente vaga. En particular, las querellas por violación de las leyes antimonopolio en Estados Unidos y en Europa se han basado en cuestiones relativas a si unidades software tales como los exploradores y los reproductores multimedia son componentes de los sistemas operativos de Microsoft o utilidades que Microsoft ha incluido simplemente para aplastar a su competencia.

Componentes de un sistema operativo

Centrémonos ahora en los componentes que conforman el dominio de un sistema operativo. Para poder llevar a cabo las acciones solicitadas por los usuarios de computadora, el sistema operativo tiene que ser capaz de comunicarse con dichos usuarios. La parte de un sistema operativo que se encarga de gestionar esa comunicación se suele denominar **interfaz de usuario**. Las antiguas interfaces de usuario, conocidas como ***shell***, se comunicaban con los usuarios mediante mensajes de texto , utilizando un teclado y la pantalla de un monitor. Los sistemas más modernas realizan esta tarea por medio de una **interfaz gráfica de usuario** (GUI, *Graphical User Interface*) en la que los objetos que hay que manipular, como por ejemplo archivos y programas, se representan de manera pictórica en la pantalla mediante iconos. Estos sistemas permiten a los usuarios ejecutar comandos mediante varios dispositivos de entrada comunes. Por ejemplo, puede utilizarse un ratón de computadora con uno o más botones, para hacer clic sobre un icono o para arrastrar iconos por la pantalla. En lugar de un ratón, los diseñadores gráficos utilizan a menudo lápices o dispositivos apuntadores de propósito especial; esos dispositivos se emplean también en diversos tipos de computadoras de mano. Más recientemente, los avances experimentados en el campo de las pantallas táctiles de granularidad fina han permitido que los usuarios puedan manipular los iconos directamente con los

Linux

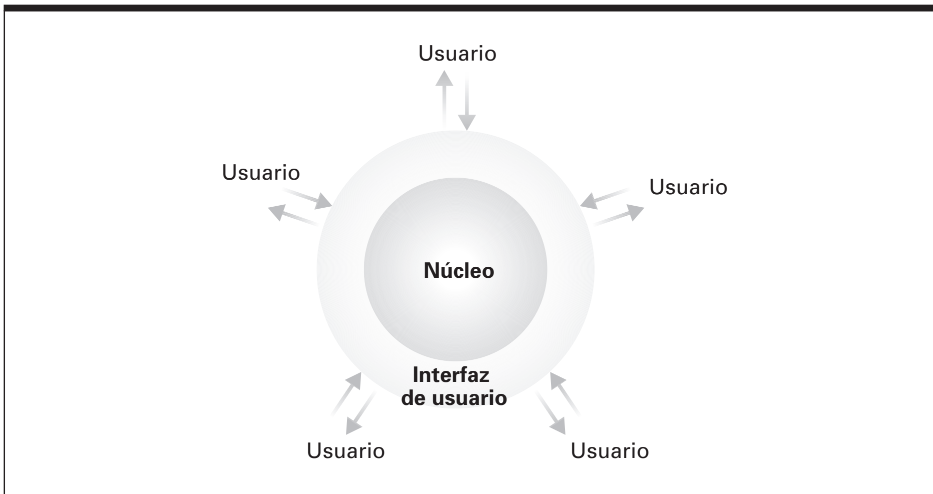
Para el entusiasta de las computadoras que desee experimentar con los componentes internos de un sistema operativo, existe un sistema denominado Linux. Linux es un sistema operativo originalmente diseñado por Linus Torvalds cuando era estudiante en la Universidad de Helsinki. Es un producto de dominio público y disponible de forma gratuita junto con su código fuente (véase el Capítulo 6) y su correspondiente documentación. Puesto que está disponible de forma gratuita en formato de código fuente, se ha hecho muy popular entre los aficionados a las computadoras, los estudiantes de sistemas operativos y los programadores en general. Además, Linux está considerado como uno de los sistemas operativos más fiables disponibles en la actualidad. Por esta razón, diversas empresas empaquetan y comercializan hoy día versiones de Linux en una forma fácilmente utilizable, y estos productos están ahora planteando todo un desafío a los sistemas operativos comerciales más implantados en el mercado. Puede obtener más información acerca de Linux en el sitio web http://www.linux.org.

dedos. Mientras que las interfaces gráficas de usuario actuales utilizan sistemas de proyección de imagen bidimensionales, en la actualidad se están investigando interfaces tridimensionales que permiten a los usuarios comunicarse con las computadoras por medio de sistemas de proyección 3D, dispositivos sensores táctiles y sistemas de reproducción de audio con sonido envolvente.

Aunque la interfaz de usuario de un sistema operativo desempeña un papel importante a la hora de establecer la funcionalidad de una máquina, esa interfaz actúa simplemente como un intermediario entre el usuario de la computadora y el verdadero corazón del sistema operativo (Figura 3.4). Esta distinción entre la interfaz de usuario y las partes internas del sistema operativo se ve enfatizada por el hecho de que algunos sistemas operativos permiten al usuario elegir entre distintas interfaces, con el fin de conseguir el tipo de interacción más confortable para ese usuario concreto. Por ejemplo, los usuarios del sistema operativo UNIX pueden elegir una shell entre los distintos disponibles, incluyendo la shell Bourne, la shell C y la shell Korn, así como una GUI denominada X11. Las primeras versiones de Microsoft Windows eran un programa de aplicación GUI que podía cargarse desde la shell de comandos del sistema operativo MS-DOS. La shell DOS cmd.exe puede todavía encontrarse como programa de utilidad en las últimas versiones de Windows, aunque los usuarios menos expertos casi nunca van a necesitar dicha interfaz. De forma similar, el sistema operativo OS X de Apple conserva la shell de utilidad Terminal que se remonta a los antecesores UNIX de dicho sistema operativo.

Un componente importante de las interfaces gráficas de usuario actuales es el **administrador de ventanas**, que asigna bloques de espacio en la pantalla, denominados ventanas, y controla qué aplicación está asociada con cada ventana. Cuando una aplicación desea mostrar algo en la pantalla, se lo notifica al administrador de ventanas y este coloca la imagen deseada en la ventana asignada a dicha aplicación. Cuando se hace clic con el ratón, es el administrador de ventanas el que calcula la ubicación del ratón en la pantalla y notifica a la aplicación apropiada cuál ha sido la acción efectuada con el ratón. Los administradores de ventanas son responsables de lo que suele denominarse "estilo" de

Figura 3.4 La interfaz de usuario actúa como intermediario entre los usuarios y el núcleo del sistema operativo.



una interfaz gráfica de usuario y la mayoría de los administradores de ventanas ofrecen una gama de opciones configurables. Los usuarios de Linux pueden incluso seleccionar qué administrador de ventanas desean utilizar, siendo las opciones más populares KDE y Gnome.

Por contraste con la interfaz de usuario de un sistema operativo, la parte interna de ese sistema se denomina **núcleo**. El núcleo de un sistema operativo contiene aquellos componentes software que realizan las funciones más básicas requeridas por el hardware de la computadora. Una de dichas unidades es el **administrador de archivos**, cuya tarea consiste en coordinar el uso de los dispositivos de almacenamiento masivo de la máquina. Para ser más precisos, el administrador de archivos mantiene un registro de todos los archivos que se encuentran en el almacenamiento masivo, incluyéndose en ese registro datos relativos a la ubicación de cada archivo, qué usuarios tienen acceso a los distintos archivos y qué partes del almacenamiento masivo están disponibles para nuevos archivos o para ampliaciones de los ya existentes. Esos registros se mantienen en el medio de almacenamiento concreto donde se ubican los archivos relacionados, de modo que cada vez que dicho medio pasa a estar en línea, el administrador de archivos puede leer esos registros y así saber lo que hay almacenado en dicho medio concreto.

Para comodidad de los usuarios de la máquina, la mayoría de los administradores de archivos permiten agrupar los archivos en una serie de conjuntos; cada uno de esos conjuntos recibe el nombre de **directorio** o **carpeta**. Esto permite al usuario organizar sus archivos de acuerdo con el propósito que tengan, colocando los archivos relacionados en un mismo directorio. Además, al permitir a los directorios contener otros directorios, que se denominan subdirectorios, puede construirse una organización jerárquica de archivos. Por ejemplo, un usuario puede crear un directorio denominado `MisDatos` que contenga subdirectorios denominados `DatosFinancieros`, `DatosMedicos` y `DatosDomesticos`. Dentro de cada uno de estos subdirectorios podría haber archivos que se correspondieran con esa categoría concreta. (Los usuarios de un sistema operativo

Windows pueden pedir al administrador de archivos que muestre la colección actual de carpetas ejecutando el programa de utilidad Explorador de Windows.)

Una cadena de directorios incluidos dentro de otros directorios se denomina **ruta de directorio**. Las rutas de directorios suelen expresarse enumerando los directorios que las componen separados por barras inclinadas. Por ejemplo, `animales/prehistoricos/dinosaurios` representaría la ruta que comienza en el directorio denominado `animales`, que pasa a través de su subdirectorio de nombre `prehistoricos` y que termina en el subdirectorio `dinosaurios`. (Para los usuarios de Windows, las barras inclinadas que se incluyen dentro de las rutas están invertidas, como por ejemplo en `animales\prehistoricos\dinosaurios`.)

Cualquier acceso a un archivo por parte de otras unidades software debe ser autorizado por el administrador de archivos. El proceso comienza solicitando al administrador de archivos que conceda acceso al archivo, para lo cual se emplea un procedimiento conocido como apertura del archivo. Si el administrador de archivos aprueba el acceso solicitado, proporciona la información necesaria para localizar y manipular el archivo.

Otro componente del núcleo es un conjunto de **controladores de dispositivo**, que son las unidades software que se comunican con las tarjetas controladoras (o en ocasiones directamente con los dispositivos periféricos) para llevar a cabo operaciones con los dispositivos periféricos conectados a la máquina. Cada controlador de dispositivo está diseñado de forma específica para su tipo de dispositivo concreto (como por ejemplo una impresora, una unidad de disco o un monitor) y traduce las solicitudes genéricas en las tareas más técnicas requeridas por el dispositivo asignado a dicho controlador. Por ejemplo, un controlador de dispositivo para una impresora contiene el software necesario para leer y decodificar la palabra de estado de esa impresora concreta, así como para coordinar el resto de los detalles del proceso de impresión. De ese modo, otros componentes software no tienen que ocuparse de todos esos detalles técnicos para imprimir un archivo. En lugar de ello, los restantes componentes pueden simplemente confiar en el software controlador del dispositivo a la hora de imprimir el archivo y dejar que ese controlador se encargue de los detalles. De esta forma, el diseño de las restantes unidades software puede ser independiente de las características distintivas de los distintos dispositivos concretos. El resultado es un sistema operativo genérico que puede personalizarse para emplear determinados dispositivos periféricos, instalando simplemente los controladores de dispositivo apropiados.

Otro componente más del núcleo de un sistema operativo es el **gestor de la memoria**, que se encarga de la tarea de coordinar el uso de la memoria principal de la máquina. Esta tarea es mínima en aquellos entornos en los que a la computadora se le pide que realice una única tarea cada vez. En estos casos, el programa encargado de realizar la tarea actual, se coloca en una posición predeterminada de la memoria principal, se ejecuta y luego se sustituye por el programa encargado de efectuar la tarea siguiente. Sin embargo, en los entornos multiusuario o multitarea en los que se le pide a la computadora que satisfaga las necesidades de muchos usuarios o programas al mismo tiempo, el trabajo del gestor de la memoria es mucho más intenso. En estos casos, muchos programas y bloques de datos tienen que coexistir en la memoria principal de manera concurrente. Por tanto, el gestor de memoria debe localizar y asignar

espacio de memoria para los distintos programas y asegurarse de que las acciones que cada programa lleva a cabo están restringidas al espacio que se le ha asignado. Además, a medida que van variando las necesidades de los diferentes programas, el gestor de la memoria debe controlar qué áreas de memoria dejan de estar ocupadas.

El trabajo del gestor de la memoria se complica más aún cuando el espacio total de memoria principal requerido por los distintos programas es superior al espacio disponible realmente en la computadora. En este caso, el gestor de memoria puede crear la ilusión de que existe espacio de memoria adicional, intercambiando los programas y los datos una y otra vez entre la memoria principal y el almacenamiento masivo (una técnica que se conoce con el nombre de **paginación**). Por ejemplo, suponga que hace falta una memoria principal de 8GB pero que la computadora solo dispone de 4GB. Para crear la ilusión de que existe un espacio de memoria de mayor tamaño, el gestor de la memoria reserva 4GB de espacio de almacenamiento en un disco magnético. Allí, escribe los patrones de bits que estarían almacenados en la memoria principal si esta tuviera realmente una capacidad de 8GB. Los datos se dividen en unidades de tamaño uniforme denominadas **páginas**, que suelen tener unos pocos KB de tamaño. Entonces, el gestor de la memoria mueve estas páginas entre la memoria principal y el almacenamiento masivo según va siendo necesario, asegurándose de que siempre estén presentes en los 4GB de memoria principal las páginas que verdaderamente se necesitan en cada momento concreto. El resultado es que la computadora es capaz de funcionar como si tuviera realmente 8GB de memoria principal. Este espacio de memoria "ficticio" de gran tamaño que se crea utilizando la técnica de paginación se conoce con el nombre de **memoria virtual**.

Otros dos componentes adicionales dentro del núcleo de un sistema operativo son el **planificador** y el **despachador**, que estudiaremos en la siguiente sección. Por ahora, simplemente diremos que en un sistema de multiprogramación, el planificador determina qué actividades son las que pueden ejecutarse, mientras que el despachador controla la asignación de tiempo a esas actividades.

Inicio del sistema operativo

Hemos visto que un sistema operativo proporciona la infraestructura software requerida por otras unidades de software, pero no hemos considerado la cuestión de cómo se inicia el sistema operativo. Esto se lleva a cabo mediante un procedimiento conocido con el nombre de **proceso de arranque** (*boot strapping* o *booting*) que es realizado por la computadora cada vez que se enciende. Es este procedimiento el que transfiere el sistema operativo desde el almacenamiento masivo (donde está almacenado de forma permanente) a la memoria principal (que está esencialmente vacía cuando se enciende la máquina por primera vez). Para entender el proceso de arranque y la razón por la que es necesario, vamos a comenzar analizando lo que sucede con el procesador de la máquina.

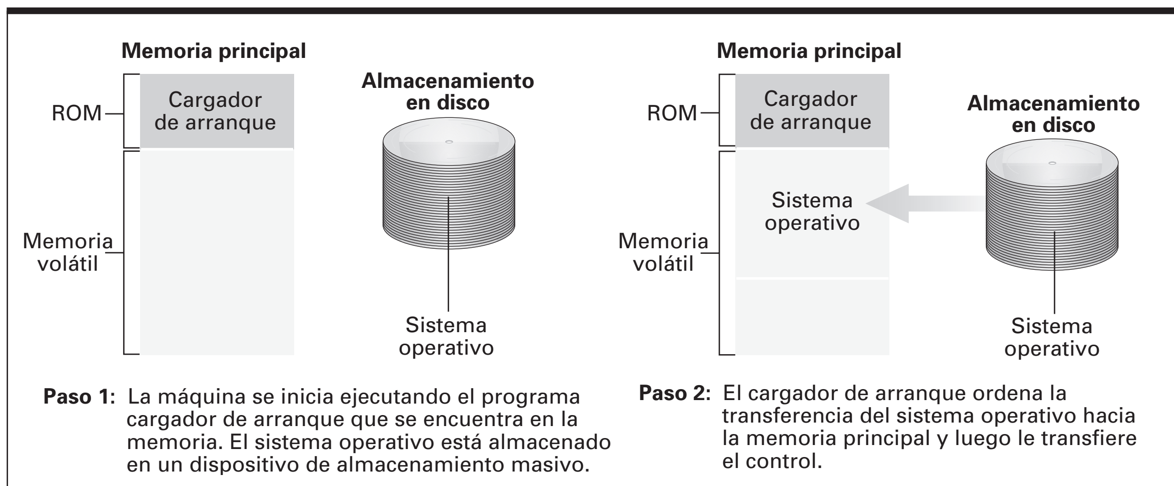
Un procesador está diseñado para que su contador de programa comience en una dirección concreta predeterminada cada vez que se inicia el procesador. Es en esa ubicación donde el procesador espera encontrar el inicio del pro-

grama que hay que ejecutar. Conceptualmente, por tanto, todo lo que necesitamos es almacenar el sistema operativo en esa ubicación. Sin embargo, por razones técnicas, la memoria principal de una computadora suele construirse mediante tecnologías volátiles, lo que quiere decir que la memoria pierde los datos almacenados en ella cada vez que se apaga la computadora. Por tanto, es preciso volver a cargar el contenido de la memoria principal cada vez que se reinicia la computadora.

En resumen, necesitamos que esté presente un programa (preferiblemente el sistema operativo) en la memoria principal cuando se enciende la computadora por primera vez, pero la memoria volátil de la memoria se borra cada vez que se apaga la máquina. Para resolver este dilema, una pequeña parte de la memoria principal de la computadora, precisamente esa parte donde el procesador espera encontrar su programa inicial, se construye con celdas especiales de memoria no volátil. Tal memoria se conoce con el nombre de **memoria de solo lectura** (ROM, *Read-Only Memory*) porque su contenido puede leerse pero no modificarse. Como analogía, podríamos pensar que el almacenar patrones de bits en ROM es como quemar pequeños fusibles (quemaríamos un fusible para almacenar un 1 y lo dejaríamos sin quemar para almacenar un 0), aunque la tecnología que se emplea es mucho más avanzada. Para ser más precisos, la mayoría de las memorias ROM que podemos encontrar en los PC actuales se construyen con tecnología de memoria flash (lo que significa que no se trata estrictamente de memoria ROM porque puede modificarse en circunstancias especiales).

En una computadora de propósito general, la ROM de la máquina almacena de forma permanente un programa denominado **cargador de arranque** (*boot loader*). Este es, por tanto, el programa que se ejecuta inicialmente cuando la máquina se enciende. Las instrucciones del cargador de arranque hacen que el procesador transfiera el sistema operativo desde una ubicación predeterminada hasta el área volátil de la memoria principal (Figura 3.5). Los cargadores de arranque modernos pueden copiar un sistema operativo en la memoria principal desde diversas ubicaciones. Por ejemplo, en los sistemas empotrados como

Figura 3.5 Proceso de arranque.



Firmware

Además del cargador de arranque, la memoria ROM de un PC contiene una serie de rutinas software que permiten realizar actividades de entrada/salida fundamentales, tales como recibir la información procedente del teclado, mostrar mensajes en la pantalla de la computadora y leer datos de los dispositivos de almacenamiento masivo. Al estar almacenado en memoria no volátil, como por ejemplo una FlashROM, este software no está impreso de manera inmutable en el silicio de la máquina (el hardware) pero tampoco se puede cambiar tan fácilmente como el resto de los programas presentes en el almacenamiento masivo (el software). Para describir esta especie de terreno intermedio se acuñó el término **firmware**. Las rutinas del firmaware pueden ser utilizadas por el cargador de arranque para realizar actividades de E/S antes de que el sistema operativo comience a funcionar. Por ejemplo, se utilizan para comunicarse con el usuario de la computadora antes de que comience el propio proceso de arranque y para informar de los errores que se produzcan durante el arranque. Entre los sistemas firmware más ampliamente utilizados se incluyen el BIOS (Basic Input/Output System, Sistema básico de entrada/salida), utilizado desde hace mucho tiempo en los PC, la interfaz EFI (Extensible Firmware Interface, Interfaz firmware ampliable), el Open Firmware de Sun (que ahora es un producto de Oracle) y el CFE (Common Firmware Environment, Entorno firmware común) utilizado en muchos dispositivos empotrados.

los teléfonos inteligentes, se copia desde una memoria flash especial (no volátil), en el caso de pequeñas estaciones de trabajo utilizadas en grandes empresas o universidades, el sistema operativo puede copiarse desde una máquina remota a través de una red. Una vez que se ha colocado el sistema operativo en la memoria principal, el cargador de arranque ordena al procesador que ejecute una instrucción de salto a dicha área de memoria. En este momento, el sistema operativo asume el mando y comienza a controlar las actividades de la máquina. El proceso global de ejecución del cargador de arranque y por tanto de inicio del sistema operativo se denomina **arranque** de la computadora.

Puede que el lector se esté preguntando por qué las computadoras de sobremesa no se proporcionan con la suficiente memoria ROM como para albergar el sistema operativo completo, de manera que no sea necesario arrancar desde el almacenamiento masivo. Aunque esta opción es factible para los sistemas empotrados que cuentan con sistemas operativos de pequeño tamaño, el dedicar grandes bloques de memoria principal en las computadoras de propósito general al almacenamiento no volátil no resulta eficiente con las tecnologías actuales. Además, los sistemas operativos para computadora sufren frecuentes actualizaciones con el fin de mantener la seguridad e incorporar controladores de dispositivo nuevos y mejorados, con el fin de poder utilizar el hardware más reciente. Aunque es posible actualizar sistemas operativos y cargadores de arranque almacenados en ROM (un proceso que a menudo se denomina **actualización del firmware**), los límites tecnológicos hacen que el almacenamiento masivo sea la opción más común en los sistemas de computadora más tradicionales.

Para terminar, conviene recalcar que comprender el proceso de arranque, así como la distinción entre sistema operativo, software de utilidad y software

de aplicación, nos permite entender la metodología global con la que operan la mayoría de las computadoras de propósito general. Cuando se enciende por primera vez una de esas máquinas, el cargador de arranque carga y activa el sistema operativo. El usuario puede hacer a partir de ese momento solicitudes al sistema operativo concernientes a los programas de aplicación o utilidades que hay que ejecutar. A medida que termina la ejecución de cada utilidad o aplicación, el usuario vuelve a estar en contacto con el sistema operativo pudiendo realizar solicitudes adicionales. El aprender a usar uno de esos sistemas es, por tanto, un proceso de dos niveles. Además de aprender los detalles de la utilidad o aplicación específicas que se desee emplear, debemos aprender lo suficiente acerca del sistema operativo de la máquina como para poder navegar entre unas aplicaciones y otras.

Cuestiones y ejercicios

1. Enumere los componentes de un sistema operativo típico y resuma el papel de cada uno de ellos en una sola frase.
2. ¿Cuál es la diferencia entre software de aplicación y software de utilidad?
3. ¿Qué es la memoria virtual?
4. Resuma el procedimiento de arranque.

3.3 Coordinación de las actividades de la máquina

En esta sección vamos a ver cómo coordina el sistema operativo la ejecución del software de aplicación, del software de utilidad y de las unidades del propio sistema operativo. Comenzaremos explicando el concepto de proceso.

El concepto de proceso

Uno de los conceptos más fundamentales de los sistemas operativos modernos es la distinción entre un programa y la actividad de ejecutar un programa. Un programa no es sino un conjunto estático de instrucciones, mientras que la ejecución del programa es una actividad dinámica cuyas propiedades cambian con el paso del tiempo. (Esta distinción es análoga a la que existe entre una partitura, que descansa inerte sobre una estantería, y la interpretación de dicha partitura por parte de un músico que ejecuta las acciones que la partitura describe.) La actividad de ejecutar un programa bajo el control del sistema operativo se conoce como **proceso**. Con el proceso está asociado el estado actual de la actividad, lo que se denomina **estado del proceso**. Este estado incluye la posición actual dentro del programa que se está ejecutando (el valor del contador de programa), así como los valores en los restantes registros del procesador y en las celdas de memoria asociadas. Simplificando, podríamos decir que el estado del proceso es una instantánea de la máquina en un momento determinado. En los diferentes momentos durante la ejecución de un programa (en los distintos instantes de un proceso), podremos observar distintas instantáneas (diferentes estados del proceso).

A diferencia de un músico, que normalmente solo trata de ejecutar una obra musical cada vez, las computadoras típicas de tiempo compartido/multitarea ejecutan muchos procesos que compiten por los recursos de la computadora. Es tarea del sistema operativo gestionar estos procesos de modo que cada uno de ellos disponga de los recursos (dispositivos periféricos, espacio en la memoria principal, acceso a archivos y acceso a un procesador) que necesita, de modo que los distintos procesos independientes no interfieran entre sí y de manera que los procesos que precisen intercambiar información puedan hacerlo.

Administración de procesos

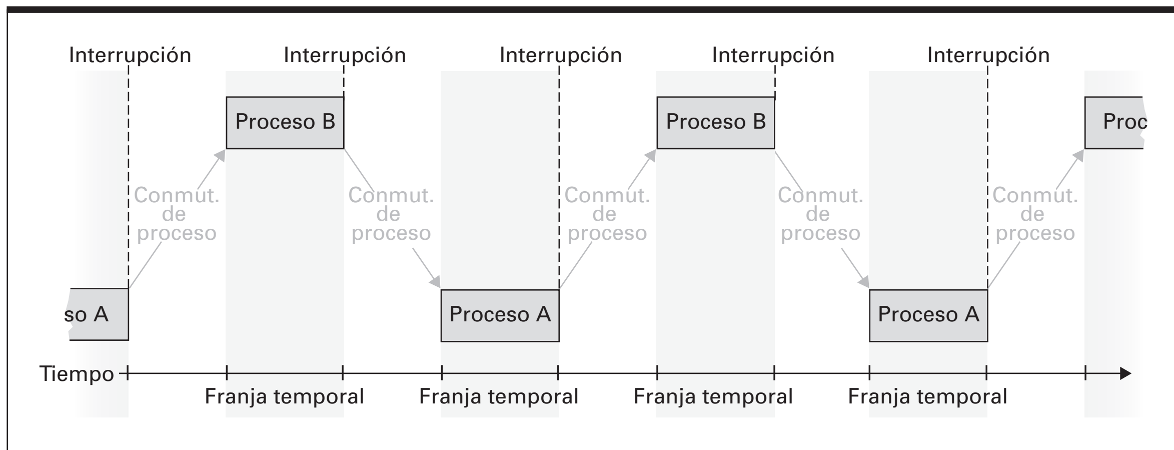
Las tareas asociadas con la coordinación de la ejecución de los procesos son gestionadas por el planificador y el despachador dentro del núcleo del sistema operativo. El planificador mantiene un registro de los procesos presentes en la computadora y se encarga de introducir nuevos procesos en dicho registro y de eliminar de él los procesos completados. Así, cuando un usuario solicita la ejecución de una aplicación, es el planificador el que añade la ejecución de dicha aplicación al conjunto de procesos actuales.

Para poder controlar todos los procesos existentes, el planificador mantiene en la memoria principal un bloque de información denominado **tabla de procesos**. Cada vez que se solicita la ejecución de un programa, el planificador crea una nueva entrada para dicho proceso en la tabla de procesos. Esta entrada contiene informaciones tales como el área de memoria asignada al proceso (obtenida del gestor de memoria), la prioridad del proceso y si el proceso está listo o a la espera. Un proceso está **listo** si se encuentra en un estado en el que puede continuar progresando; estará **a la espera** si su progreso está actualmete detenido hasta que tenga lugar algún suceso externo, como la terminación de una operación de escritura en almacenamiento masivo, la pulsación de una tecla del teclado o la llegada de un mensaje enviado por otro proceso.

El despachador es el componente del núcleo que supervisa la ejecución de los procesos planificados. En un sistema de tiempo compartido/multitarea, esta tarea se lleva a cabo mediante **multiprogramación**; es decir, dividiendo el tiempo en cortos segmentos, cada uno de los cuales se denomina **franja temporal** (que típicamente dura unos milisegundos o unos microsegundos), y luego conmutando la atención del procesador entre los distintos procesos, a medida que se permite que cada uno de ellos se ejecute durante una franja temporal (Figura 3.6). El procedimiento de cambiar de un proceso a otro se denomina **conmutación de procesos** (o **cambio de contexto**).

Cada vez que el despachador concede una franja temporal a un proceso, inicia un circuito temporizador que indicará el final de la franja temporal, generando una señal conocida como **interrupción**. El procesador reacciona a esta señal de interrupción de forma bastante similar a como nosotros reaccionamos cuando nos interrumpen en mitad de una tarea: dejamos de hacer lo que estuviéramos haciendo, anotamos en qué punto de la tarea nos encontramos (para poder volver a ella posteriormente) y atendemos a la entidad que nos está interrumpiendo. Cuando el procesador recibe una señal de interrupción, completa su ciclo de máquina actual, anota su posición dentro del proceso actual y comienza a ejecutar un programa, denominado **rutina de tratamiento de**

Figura 3.6 Multiprogramación entre el proceso A y el proceso B.



interrupciones, que está almacenado en una posición predeterminada de la memoria principal. Esta rutina de tratamiento de interrupciones forma parte del despachador y se encarga de describir cómo debe responder el despachador a la señal de interrupción.

Por tanto, el efecto de la señal de interrupción es el de desalojar al proceso actual y transferir de nuevo el control al despachador. En ese momento, el despachador selecciona en la tabla de procesos aquel proceso que tenga la mayor prioridad entre todos los procesos que estén listos (según haya determinado el planificador); a continuación, el despachador reinicia el circuito de temporización y permite al proceso seleccionado que comience con su franja temporal.

Interrupciones

El uso de interrupciones para la terminación de franjas temporales como se describe en el texto, es tan solo una de las múltiples aplicaciones del sistema de interrupciones de una computadora. Existen muchas situaciones distintas en las que se genera una señal de interrupción, teniendo cada una de las posibles señales de interrupción su propia rutina de tratamiento de la misma. De hecho, las interrupciones proporcionan una importante herramienta para la coordinación de las acciones de una computadora con su entorno. Por ejemplo, tanto hacer clic con el ratón como pulsar una tecla del teclado generan señales de interrupción que hacen que el procesador deje a un lado su actividad actual y atienda a la causa de la interrupción.

Para gestionar la tarea de reconocer y responder a las interrupciones entrantes, a las distintas señales de interrupción se les asignan prioridades, de modo que se pueda atender primero a las tareas más importantes. La interrupción de más alta prioridad suele estar asociada con un fallo de la alimentación eléctrica. Dicha señal de interrupción se genera si de manera inesperada se corta el suministro de energía eléctrica a la computadora. La rutina de interrupción asociada dirige al procesador a través de una serie de tareas básicas de “salvaguarda” durante los escasos milisegundos disponibles antes de que el nivel de tensión caiga por debajo del nivel mínimo que garantiza el correcto funcionamiento de la computadora.

El aspecto clave para el buen funcionamiento de un sistema de multiprogramación es la capacidad de detener un proceso y reanudarlo posteriormente. Si nos interrumpen mientras estamos leyendo un libro, nuestra capacidad de continuar leyendo posteriormente dependerá de nuestra capacidad de recordar el punto del libro en el que nos encontrábamos, así como la información que habíamos acumulado hasta ese punto. En resumen, debemos poder crear de nuevo el entorno que existía inmediatamente antes de la interrupción.

En el caso de un proceso, el entorno que hay que recrear es el estado del proceso que, como ya hemos mencionado, incluye el valor del contador de programa, así como el contenido de los registros y de las celdas de memoria pertinentes. Los procesadores diseñados para sistemas de multiprogramación incorporan la tarea de guardar esta información como parte de la reacción del procesador a la señal de interrupción. Estos procesadores también tienden a disponer de instrucciones del lenguaje máquina específicamente pensadas para recargar un estado previamente guardado. Dichas características simplifican la tarea del despachador cuando se realiza una conmutación de procesos y constituyen un ejemplo de cómo el diseño de los procesadores modernos está influenciado por las necesidades de los sistema operativos actuales.

Para finalizar, es preciso observar que el uso de técnicas de multiprogramación permite incrementar la eficiencia global de una máquina. Esto puede resultar anti-intuitivo, ya que la conmutación entre procesos requerida por la multiprogramación introduce una cierta sobrecarga de carácter meramente administrativo. Sin embargo, sin la multiprogramación cada proceso se ejecutaría hasta completarse antes de que el siguiente proceso pudiera comenzar, lo que significa que todo el tiempo que el proceso invierte en esperar a que los dispositivos periféricos completen tareas o a que un usuario haga su siguiente solicitud se desperdicia. La multiprogramación permite que estos tiempos muertos sean aprovechados por otros procesos. Por ejemplo, si un proceso ejecuta una solicitud de E/S, como por ejemplo una solicitud para extraer datos de un disco magnético, el planificador actualizará la tabla de procesos para reflejar el hecho de que el proceso está esperando a que tenga lugar un suceso externo. A su vez, el despachador dejará de conceder franjas temporales a dicho proceso. Posteriormente (quizá varios centenares de milisegundos después), cuando se haya completado la solicitud de E/S, el planificador actualizará la tabla de procesos para indicar que el proceso está listo, con lo que ese proceso volverá a competir de nuevo por el uso de las franjas temporales. En resumen, se podrá ir progresando en la realización de estas tareas mientras que se completa la solicitud de E/S, de modo que todo el conjunto de tareas se completará en menos tiempo que si se ejecutaran de manera secuencial.

Cuestiones y ejercicios

1. Resuma la diferencia entre un programa y un proceso.
2. Resuma los pasos que lleva a cabo el procesador cuando se produce una interrupción.
3. En un sistema de multiprogramación, ¿cómo se puede hacer que los procesos de alta prioridad se ejecuten más rápidamente que los demás?

4. Si cada franja temporal de un sistema de multiprogramación dura 50 milisegundos y cada cambio de contexto requiere como máximo un microsegundo, ¿a cuántos procesos puede dar servicio la máquina en un único segundo?
5. Si cada proceso utiliza completamente su franja temporal en la máquina de la Cuestión 4, ¿qué fracción del tiempo de la máquina se invierte en ejecutar realmente procesos? ¿ Cuál sería dicha fracción si cada proceso ejecutara una solicitud de E/S después de transcurrido un solo microsegundo de su franja temporal?

3.4 Gestión de la competición entre procesos

Una tarea importante de un sistema operativo es la asignación de los recursos de la máquina a los procesos del sistema. Aquí, utilizamos el término ***recurso*** en un sentido amplio, para incluir tanto los dispositivos periféricos de la máquina como las funcionalidades de la propia máquina. El administrador de archivos asigna el acceso a los archivos, asigna el espacio de almacenamiento masivo para la construcción de archivos nuevos; el gestor de memoria asigna espacio de memoria; el planificador asigna el espacio dentro de la tabla de procesos y el despachador asigna las franjas temporales de procesamiento. Al igual que sucede con muchos otros problemas dentro de los sistemas de computación, esta tarea de asignación puede parecer simple a primera vista. Sin embargo, por debajo de la superficie se encuentran diversas sutilezas que pueden hacer que las operaciones sean incorrectas en un sistema mal diseñado. Recuerde que una máquina no piensa por sí misma, sino que se limita a seguir las instrucciones que le damos. En consecuencia, para construir sistemas operativos fiables es necesario desarrollar algoritmos que tengan en cuenta todas las posibles contingencias, independientemente de lo improbables que puedan parecer.

Semáforos

Imaginemos un sistema operativo de tiempo compartido/multitarea que controla las actividades de una computadora que cuenta con una única impresora. Si un proceso necesita imprimir sus resultados, tendrá que solicitar que el sistema operativo le de acceso al controlador de dispositivo de la impresora. En ese momento el sistema operativo deberá decidir si aprueba dicha solicitud, dependiendo de si la impresora ya está siendo utilizada por otro proceso. Si no hay ningún otro proceso usándola, el sistema operativo debe aprobar la solicitud y permitir al proceso continuar; en caso contrario, el sistema operativo debe rechazar la solicitud y, quizá, clasificar al proceso como un proceso en espera hasta que la impresora esté disponible. Después de todo, si concediéramos a dos procesos acceso simultáneo a la impresora de la computadora, el resultado sería inútil para ambos procesos.

Para controlar el acceso a la impresora, el sistema operativo debe anotar si la impresora ya ha sido asignada. Una forma de hacer esto sería utilizando un indicador, que en este contexto sería un bit en la memoria cuyos estados

se suelen denominar ***activado (set)*** y ***desactivado (clear),*** en lugar de 1 y 0. Un indicador desactivado (valor 0) señala que la impresora está disponible mientras que un indicador activado (valor 1) nos informa de que la impresora está actualmente asignada. A primera vista, este enfoque parece correcto: el sistema operativo se limita a comprobar el indicador cada vez que le llega una solicitud de acceso a la impresora. Si el indicador está desactivado, se aprueba la solicitud y el sistema operativo activa el indicador. Si el indicador está activado, el sistema operativo obliga al proceso solicitante a esperar. Cada vez que un proceso termina de utilizar la impresora, el sistema operativo asigna la impresora a uno de los procesos en espera o, si no hay ninguno, se limita a desactivar el indicador.

Sin embargo, este sistema indicador tan simple tiene un problema. La tarea de verificar el estado y, posiblemente, configurar el indicador puede requerir varias instrucciones en la máquina (es preciso extraer el valor del indicador de la memoria principal, hay que manipularlo en el procesador y finalmente hay que volver a almacenarlo en la memoria). Por tanto, es posible que una tarea sea interrumpida después de haber detectado que el indicador está desactivado, pero antes de haberlo activado. En particular, suponga que la impresora está actualmente disponible y que un proceso ha solicitado utilizarla. Se extrae el indicador de la memoria principal y comprobamos que está desactivado, lo que indica que la impresora está disponible. Sin embargo, en ese momento el proceso es interrumpido y otro proceso distinto comienza a hacer uso de su franja temporal. Imagine que ese segundo proceso solicita también utilizar la impresora. De nuevo, se extrae el indicador de la memoria principal y se comprueba que sigue estando desactivado, porque el proceso anterior fue interrumpido antes de que el sistema operativo tuviera tiempo de configurar el indicador como activado en la memoria principal. En consecuencia, el sistema operativo permite al segundo proceso comenzar a emplear la impresora. Posteriormente, el proceso original reanuda su ejecución en el punto en el que la había dejado, que es exactamente el punto inmediatamente posterior al momento en el que el sistema operativo determinó que el indicador estaba desactivado. Como resultado, el sistema operativo continúa con las operaciones previstas, activando el indicador en la memoria principal y concediendo al proceso original acceso a la impresora. El resultado final es que ahora habrá dos procesos utilizando la impresora.

Administrador de tareas de Microsoft

Podemos tratar de entender parte de las actividades internas de un sistema operativo Microsoft Windows ejecutando el programa de utilidad denominado Administrador de tareas (pulse las teclas Ctrl, Alt y Supr simultáneamente). En particular, seleccionando la pestaña Procesos en la ventana del Administrador de tareas podemos acceder a la tabla de procesos. He aquí un experimento que puede tratar de realizar: examine la tabla de procesos antes de activar ningún programa de aplicación (es posible que le sorprenda ver que hay tantos procesos ya en la tabla. Esos procesos son necesarios para el funcionamiento básico del sistema). Ahora active una aplicación y compruebe cómo se habrá añadido un proceso adicional a la tabla. También podrá ver cuánto espacio de memoria se ha asignado a dicho proceso.

La solución a este problema consiste en garantizar que la tarea de comprobar y posiblemente configurar el indicador se complete sin interrupciones. Una posibilidad es usar las instrucciones de desactivación y activación de interrupciones que se proporcionan en la mayoría de los lenguajes máquina. Al ejecutarse, una instrucción de desactivación de interrupciones hace que se bloqueen las interrupciones futuras, mientras que una instrucción de activación de interrupciones hace que el procesador comience de nuevo a responder a las señales de interrupción. Así, si el sistema operativo inicia su rutina de comprobación del indicador con una instrucción de desactivación de las interrupciones y termina esa rutina con una instrucción de activación de las interrupciones, ninguna otra actividad podrá interrumpir a esa rutina después de que esta haya comenzado.

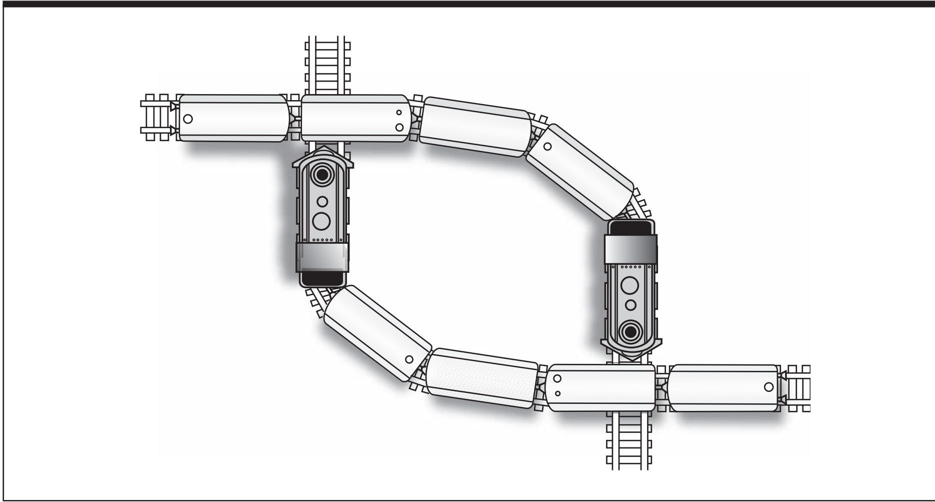
Otra solución sería emplear la instrucción de **comprobación y configuración** (*test-and-set*) disponible en muchos lenguajes máquina. Esta instrucción hace que el procesador lea el valor de un indicador, anote el valor recibido y luego configure el indicador, todo ello dentro de una misma instrucción de lenguaje máquina. La ventaja es que, como el procesador siempre completa la instrucción en curso antes de atender a ninguna otra interrupción, la tarea de comprobar y configurar el indicador no puede partirse cuando se implementa como una única instrucción.

Un indicador adecuadamente implementado, como por ejemplo de la forma que acabamos de describir, se denomina **semáforo**, en referencia a las señales de las vías férreas y de las carreteras que se utilizan para controlar el acceso a las distintas secciones de las mismas. De hecho, los semáforos se emplean en los sistemas software de forma bastante similar a como se utilizan sus equivalentes en las líneas férreas. El equivalente a una sección de una vía que solo puede contener un tren al mismo tiempo sería una secuencia de instrucciones que solo debe ser ejecutada por un proceso cada vez. A una secuencia de instrucciones de este tipo se la denomina **región crítica**. El requisito de que solo se permita a un proceso cada vez ejecutar una región crítica se conoce con el nombre de **exclusión mutua**. Resumiendo, una forma común de conseguir la exclusión mutua para una cierta región crítica consiste en proteger la región crítica mediante un semáforo. Para entrar en la región crítica, el proceso debe encontrarse primero el semáforo desactivado y luego activar el semáforo antes de entrar en dicha región crítica; después, al salir de la región crítica, el proceso debe desactivar el semáforo. Si un semáforo está activado, el proceso que está intentando entrar en la región crítica deberá esperar hasta que el semáforo se desactive.

Interbloqueo

Otro problema que puede surgir durante la asignación de recursos es lo que se conoce con el nombre de **interbloqueo**, que es la condición en la que dos o más procesos están impedidos de progresar debido a que cada uno de ellos está esperando por un cierto recurso que está asignado al otro. Por ejemplo, un proceso puede tener acceso a la impresora de la computadora pero estar esperando acceder al reproductor de CD, mientras que otro proceso puede tener acceso al reproductor de CD pero estar esperando por la impresora. Otro ejemplo es el que a veces surge en sistemas en los que se permite a los procesos crear nuevos

Figura 3.7 Un interbloqueo resultante de la competencia por cruces de vías férreas que no son compartibles.



procesos (una acción denominada **bifurcación**, *forking*, en lenguaje UNIX) para realizar subtareas. Si el planificador ya no dispone de espacio libre dentro de la tabla de procesos y cada proceso del sistema tiene que crear un proceso adicional antes de completar la tarea que tiene asignada, entonces ninguno de los procesos podrá continuar. Dichas condiciones, al igual que las que surgen en otros entornos (Figura 3.7), puede degradar seriamente el rendimiento de un sistema.

El análisis de las situaciones de interbloqueo revela que estas situaciones no pueden producirse a menos que se satisfagan las siguientes tres condiciones:

1. Hay una competencia por el acceso a recursos no compartibles.
2. Los recursos se solicitan de manera parcial; es decir, habiendo recibido algunos recursos, un proceso puede volver posteriormente a solicitar recursos adicionales.
3. Una vez que un recurso ha sido asignado, esa asignación no puede ser anulada.

El objeto de enunciar estas condiciones es darnos cuenta de que el problema del interbloqueo puede afrontarse consiguiendo que cualquiera de estas tres condiciones no se cumpla. Las técnicas dirigidas a la tercera de estas condiciones caen dentro de la categoría que se conoce con el nombre de esquemas de detección y corrección de interbloqueos. En estos casos, la posibilidad de que se produzca un interbloqueo se considera tan remota que no se intenta hacer ningún esfuerzo para evitar el problema. En lugar de ello, la solución adoptada consiste en detectar el interbloqueo cuando se produce y luego corregirlo anulando algunas de las asignaciones de recursos realizadas. Nuestro ejemplo de la tabla de procesos llena podría caer dentro de esta categoría. Si se produjera un interbloqueo debido a que la tabla de procesos está llena, una serie de rutinas del sistema operativo (o quizá un administrador humano, utilizando sus poderes de "superusuario") puede eliminar [el término técnico es **matar**, (*kill*)] algunos de los procesos. Esto hace que se libere espacio dentro de la tabla de

procesos, rompiendo el interbloqueo y permitiendo a los restantes procesos continuar con sus tareas.

Las técnicas que tratan de lidiar con las dos primeras condiciones se conocen con el nombre de esquemas de evitación de interbloqueos. Una de ellas, por ejemplo, trata de que no se cumpla la segunda condición, exigiendo que cada proceso solicite todos los recursos necesarios a la vez. Otro esquema trata de conseguir que no se cumpla la primera condición no eliminando directamente la competición, sino convirtiendo los recursos no compartibles en recursos que sí se puedan compartir. Por ejemplo, suponga que el recurso en cuestión es una impresora y que son varios los procesos que necesitan utilizarla. Cada vez que un proceso solicite la impresora, el sistema operativo podría conceder dicha solicitud; pero en lugar de conectar el proceso a la impresora, lo que el sistema operativo haría sería conectarlo a un controlador de dispositivo que guardara en el almacenamiento masivo la información que hay que imprimir, en lugar de enviarla a la impresora directamente. De ese modo, cada uno de los procesos pensando que tiene acceso a la impresora, podría ejecutarse en forma normal. Posteriormente, cuando la impresora estuviera disponible, el sistema operativo podía transferir los datos desde el almacenamiento masivo a la impresora. De esta forma, el sistema operativo estaría haciendo que ese recurso no compartible parezca ser compartible, creando la ilusión de que hay más de una impresora. Esta técnica de guardar los datos de salida para enviarlos en un momento posterior más conveniente se denomina **spooling**.

Hemos introducido el ***spooling*** como una técnica que sirve para conceder acceso a varios procesos a un recurso común, lo cual es un tema que tiene múltiples variaciones. Por ejemplo, un administrador de archivos podría autorizar que varios procesos accedieran al mismo archivo, si esos procesos están mera-

Sistemas operativos multinúcleo

Los sistemas de tiempo compartido/multitarea proporcionan la ilusión de estar ejecutando muchos procesos a la vez conmutando rápidamente entre unas franjas temporales y otras, con mucha mayor velocidad de lo que los seres humanos somos capaces de percibir. Los sistemas modernos continúan realizando la multitarea de esta forma, pero además, los procesadores multinúcleo más recientes son verdaderamente capaces de realizar dos, cuatro o muchos más procesos simultáneamente. A diferencia de un grupo de computadoras de un solo núcleo que estuvieran trabajando conjuntamente, una máquina multinúcleo contiene varios procesadores independientes (que en este caso se denominan núcleos) que comparten los periféricos, la memoria y otros recursos de la computadora. Para un sistema operativo multinúcleo, esto quiere decir que el despachador y el planificador deben considerar qué procesos hay que ejecutar en cada uno de los núcleos disponibles. Con diferentes procesos ejecutándose en distintos núcleos, la gestión de la competencia entre procesos se vuelve más complicada, porque la desactivación de las interrupciones en todos los núcleos cuando uno de ellos necesita entrar en una región crítica sería muy ineficiente. La Ciencia de la computación tiene muchas áreas activas de investigación relacionadas con la construcción de mecanismos para sistemas operativos mejor adaptados al nuevo mundo de los procesadores multinúcleo.

mente leyendo datos de él, pero podrían aparecer conflictos si hay más de un proceso tratando de modificar un archivo al mismo tiempo. Por tanto, un administrador de archivos podría asignar el acceso a los archivos de acuerdo con las necesidades de los procesos, permitiendo que varios procesos tengan acceso de lectura y que solo un proceso tenga acceso de escritura. Otros sistemas pueden dividir el archivo en fragmentos, de modo que los diferentes procesos puedan modificar distintas partes del archivo concurrentemente. Sin embargo, cada una de estas técnicas presenta sutilezas que hay que resolver para obtener un sistema fiable. Por ejemplo, ¿cómo podría notificarse a esos procesos que solo tienen acceso de lectura a un archivo que un proceso con acceso de escritura ha modificado el archivo?

Cuestiones y ejercicios

1. Suponga que los procesos A y B están compartiendo tiempo en la misma máquina y que cada uno de ellos necesita el mismo recurso no compartible durante cortos periodos de tiempo (por ejemplo, cada proceso puede estar imprimiendo una serie de informes independientes de pequeña longitud). Cada proceso puede entonces adquirir repetidamente el recurso, liberarlo y volverlo a solicitar. ¿Qué desventaja tendría controlar el acceso al recurso de la forma siguiente?:

   > Comenzamos asignando a un indicador el valor 0. Si el proceso A solicita el recurso y el indicador es 0, se aprueba la solicitud. En caso contrario, se obliga al proceso A a esperar. Si el proceso B solicita el recurso y el indicador es 1, se concede la solicitud. En caso contrario, se obliga al proceso B a esperar. Cada vez que el proceso A finaliza con el recurso, se cambia el indicador a 1. Cada vez que el proceso B termina con el recurso, se cambia el indicador a 0.

2. Suponga que una carretera de dos sentidos converge a un único carril para atravesar un túnel. Para coordinar el uso del túnel, se ha instalado el siguiente sistema de señalización:

   > Un vehículo que entre por cualquiera de los extremos del túnel hace que se enciendan las señales luminosas de color rojo situadas encima de las dos entradas del túnel. Cuando el vehículo sale del túnel, las señales luminosas se apagan. Si un vehículo que se está aproximando al túnel se encuentra con una señal luminosa roja encendida, espera hasta que se apaga dicha señal antes de entrar en el túnel.

   ¿Cuál es el fallo de este sistema?

3. Suponga que se han propuesto las siguientes soluciones para eliminar el interbloqueo que tiene lugar en un puente de un solo carril, cuando dos vehículos se encuentran de frente en él. Identifique cuál de las condiciones mencionadas en el texto para la aparición de interbloqueos se elimina con cada una de las siguientes soluciones.

   a. No permitir que un vehículo entre en el puente hasta que el puente esté vacío.

b. Si dos vehículos se encuentran de frente en el puente, hacer que uno de ellos retroceda.
c. Añadir un segundo carril al puente.

4. Suponga que representamos cada proceso en un sistema de multiprogramación y que dibujamos una flecha desde un punto a otro si el proceso representado por el primer punto está esperando por un recurso (no compartible) que está siendo utilizado por el segundo. Los matemáticos llaman al dibujo resultante **grafo dirigido**. ¿Qué propiedad del grafo dirigido sería equivalente a la aparición de un interbloqueo en el sistema?

3.5 Seguridad

Puesto que el sistema operativo supervisa las actividades de una computadora, es natural que desempeñe también un papel vital en el mantenimiento de la seguridad. En un sentido amplio, esta responsabilidad se manifiesta de múltiples formas, una de las cuales es la fiabilidad. Si un fallo del administrador de archivos provoca la pérdida de parte de un archivo, entonces ese archivo no estaba seguro. Si un defecto en el despachador conduce a un fallo del sistema, provocando la pérdida de todos los datos introducidos a través del teclado en la última hora, podríamos decir que nuestro trabajo no estaba seguro. Por tanto, la seguridad de una computadora requiere un sistema operativo bien diseñado y en el que se pueda confiar.

El desarrollo de software fiable no es un campo restringido al campo de los sistemas operativos. Por el contrario, abarca todo el espectro de desarrollo de software y constituye el campo de las Ciencias de la computación conocido con el nombre de Ingeniería del software, que veremos en el Capítulo 7. En esta sección nos vamos a centrar entonces en aquellos problemas de seguridad que están más estrechamente relacionados con las especifidades de los sistemas operativos.

Ataques desde el exterior

Una tarea de gran importancia de la que se encargan los sistemas operativos es la de proteger los recursos de la computadora frente a los accesos por parte de personal no autorizado. En el caso de computadoras utilizadas por varias personas, la solución habitual consiste en definir "cuentas" para los diversos usuarios, siendo una cuenta, esencialmente, un registro dentro del sistema operativo que contiene datos tales como el nombre del usuario, la contraseña y los privilegios que hay que conceder a dicho usuario. El sistema operativo puede utilizar entonces esta información durante cada proceso de **inicio de sesión** (*login*), con el fin de controlar el acceso al sistema. Un inicio de sesión es una secuencia de transacciones en la que el usuario establece el contacto inicial con el sistema operativo de la computadora.

Las cuentas son definidas por una persona conocida como **superusuario** o **administrador**. Esta persona obtiene un acceso altamente privilegiado al sis-

tema operativo, identificándose ante él como administrador (usualmente mediante su nombre y su contraseña) durante el proceso de inicio de sesión. Una vez establecido este contacto, el administrador puede alterar las configuraciones dentro del sistema operativo, modificar los paquetes software críticos, ajustar los privilegios concedidos a otros usuarios y realizar diversas otras actividades de mantenimiento para las que los usuarios normales no tienen autorización.

Desde esta atalaya privilegiada, el administrador también es capaz de monitorizar las actividades dentro del sistema de computación, en un esfuerzo por detectar los comportamientos destructivos, ya sean maliciosas o accidentales. Como ayuda para esta tarea, se han desarrollado numerosas utilidades software, denominadas **software de auditoría**, que registran y luego analizan las actividades que tienen lugar dentro de la computadora. En particular, el software de auditoría puede revelar una catarata de intentos de inicio de sesión mediante contraseñas incorrectas, que indican que un usuario no autorizado puede estar intentando tener acceso a la computadora. El software de auditoría también puede identificar actividades dentro de una cuenta de usuario que no se ajustan al comportamiento pasado de dicho usuario, lo que puede ser indicativo de que un usuario no autorizado ha tenido acceso a dicha cuenta. (Resulta extraño que un usuario que normalmente solo utiliza software de procesamiento de textos y de hoja de cálculo comience de repente a acceder a aplicaciones software muy técnicas o intente ejecutar paquetes de utilidad que caen fuera de los privilegios de dicho usuario.)

Otro culpable para el que los sistemas de auditoría están diseñados es tratar de detectar la presencia de **software espía** (*sniffing*), software que, cuando se ejecuta en una computadora, registra las actividades que en ella se realizan y posteriormente informa de las mismas a un intruso potencial. Un ejemplo bastante tradicional y bien conocido son los programas que simulan el proceso de inicio de sesión del propio sistema operativo. Puede emplearse ese tipo de programas para engañar a los usuarios autorizados, haciéndoles creer que se están comunicando con el sistema operativo, mientras que lo que en realidad están haciendo es suministrar sus nombres y contraseñas a un impostor.

Teniendo en cuenta toda la complejidad técnica asociada con la seguridad en los sistemas de computación, resulta sorprendente para muchas personas saber que uno de los principales obstáculos para dotar de seguridad a las computadoras son los descuidos de los propios usuarios. Los usuarios eligen a veces contraseñas que son relativamente fáciles de adivinar (como por ejemplo nombres y fechas), comparten sus contraseñas con los amigos, se olvidan de cambiar sus contraseñas de manera periódica, someten a los dispositivos de almacenamiento masivo fuera de línea a una degradación potencial transfiriéndolos una y otra vez entre máquinas e introducen en el sistema software no aprobado que puede subvertir la seguridad del sistema. Para lidiar con problemas como estos, la mayoría de las instituciones que disponen de grandes instalaciones de computadoras adoptan e imponen una serie de políticas que catalogan los requisitos que los usuarios deben cumplir y sus responsabilidades.

Ataques desde el interior

Una vez que un intruso (o quizá un usuario autorizado con intenciones maliciosas) obtiene un acceso a una computadora, el siguiente paso consiste en

explorar el sistema, buscando información de interés o lugares en los que insertar software destructivo. Este es un proceso muy sencillo si el intruso ha obtenido acceso a la cuenta del administrador, razón por la cual la contraseña del administrador se guarda tan celosamente. Sin embargo, si el acceso se produce a través de la cuenta de un usuario normal, entonces es necesario engañar al sistema operativo, para que permita al intruso ir más allá de los privilegios concedidos a ese usuario. Por ejemplo, el intruso puede intentar engañar al gestor de memoria, para que este permita a un proceso acceder a celdas de la memoria principal situadas fuera de su área asignada. O bien, el intruso puede tratar de engañar al administrador de archivos, para que extraiga archivos cuyo acceso debería ser denegado.

Hoy día, los procesadores han sido mejorados con una serie de características diseñadas para tratar de desbaratar esos intentos de intrusión. Por ejemplo, considere la necesidad de restringir un proceso al área de la memoria principal que el gestor de memoria le ha asignado. Sin dichas restricciones, un proceso podría borrar el sistema operativo de la memoria principal y tomar el control de la computadora. Para contrarrestar esos intentos, los procesadores diseñados para sistemas de multiprogramación suelen contener registros de propósito especial en los que el sistema operativo puede almacenar los límites superior e inferior del área de memoria asignada a un proceso. Entonces, mientras está ejecutando el proceso, el procesador compara cada referencia a memoria con dichos registros, para garantizar que la referencia se encuentre dentro de los límites designados. Si se ve que la referencia cae fuera del área designada para el proceso, el procesador transfiere automáticamente el control al sistema operativo (realizando una secuencia de interrupción), de modo que el sistema operativo pueda llevar a cabo las acciones apropiadas.

Escondido en este ejemplo se encuentra un problema sutil pero importante. Sin funciones de seguridad adicionales, un proceso podría continuar obteniendo acceso a celdas de memoria situadas fuera de su área designada, simplemente cambiando los registros de propósito especial que contienen sus límites de memoria. Es decir, un proceso que deseara acceder a memoria adicional podría simplemente incrementar el valor del registro donde está almacenado el límite de memoria superior y luego utilizar ese espacio de memoria adicional sin la aprobación del sistema operativo.

Para protegerse frente a tales acciones, los procesadores para sistemas de multiprogramación están diseñados para operar en uno de dos posibles **niveles de privilegio**; llamaremos a uno de ellos "modo privilegiado" y al otro "modo no privilegiado". En el modo privilegiado, el procesador es capaz de ejecutar todas las instrucciones de su lenguaje máquina. Sin embargo, cuando opera en modo no privilegiado, la lista de instrucciones aceptables está limitada. Las instrucciones que solo están disponibles en el modo privilegiado se denominan **instrucciones privilegiadas**. Como ejemplos típicos de instrucciones privilegiadas podríamos citar las instrucciones que modifican el contenido de los registros que indican los límites de memoria y las instrucciones que cambian el modo de privilegio actual del procesador. Cualquier intento de ejecutar una instrucción privilegiada mientras que el procesador se encuentra en modo no privilegiado provoca una interrupción. Esta interrupción hace pasar al procesador al modo privilegiado y transfiere el control a una rutina de tratamiento de interrupciones situada dentro del propio sistema operativo.

Cuando se enciende por primera vez, el procesador opera en modo privilegiado. De este modo, cuando el sistema operativo se inicia al final del proceso de arranque, todas las instrucciones son ejecutables. Sin embargo, cada vez que el sistema operativo permite a un proceso comenzar con una franja temporal, conmuta al procesador a modo no privilegiado ejecutando una instrucción "cambio del modo de privilegio". A su vez, el sistema operativo será notificado si el proceso intenta ejecutar una instrucción privilegiada, y de ese modo el sistema operativo será capaz de mantener la integridad de la computadora.

Las instrucciones privilegiadas y el control de los niveles de privilegio es la herramienta principal que los sistemas operativos tienen a su disposición para poder mantener la seguridad. Sin embargo, el uso de estas herramientas es un componente complejo del diseño de un sistema operativo y en los sistemas actuales continúan encontrándose errores. Un único fallo en el control del nivel de privilegio puede abrir la puerta a numerosos desastres provocados por programadores maliciosos o por errores de programación involuntarios. Si se permite a un proceso modificar el temporizador que controla el sistema de multiprogramación de la máquina, dicho proceso puede ampliar su franja temporal y hacerse con la máquina. Si se permite a un proceso acceder directamente a los dispositivos periféricos, entonces podrá leer archivos sin la supervisión del administrador de archivos del sistema. Si se permite a un proceso acceder a celdas de memoria situadas fuera de su área asignada, podrá leer e incluso modificar los datos que estén siendo usados por otros procesos. Por tanto, el mantenimiento de la seguridad continúa siendo una de las tareas más importantes de un administrador, así como uno de los principales objetivos en el diseño de los sistemas operativos.

Cuestiones y ejercicios

1. Proporcione algunos ejemplos de elecciones inadecuadas de contraseñas y explique por qué son inadecuadas.
2. Los procesadores de la serie Pentium de Intel proporcionan cuatro niveles de privilegio. ¿Por qué los diseñadores de procesadores podrían querer proporcionar cuatro niveles en lugar de tres o cinco?
3. Si un proceso de un sistema de multiprogramación pudiera acceder a celdas de memoria situadas fuera de su área asignada, ¿cómo podría hacerse con el control de la máquina?

Problemas de repaso

(Los problemas con asterisco están asociados con las secciones opcionales.)

1. Enumere cuatro actividades de un sistema operativo típico.
2. Resuma la diferencia entre procesamiento por lotes y procesamiento interactivo.
3. Suponga que colocamos tres elementos R, S y T en una cola en dicho orden. Elimina-

mos entonces un elemento de la cola antes de añadir a esta un cuarto elemento, X. A continuación, eliminamos un elemento de la cola, introducimos en esta los elementos Y y Z, y luego vaciamos la cola eliminando un elemento cada vez. Enumere todos los elementos en el orden en que han sido extraídos de la cola.

4. ¿Cuál es la diferencia entre los sistemas empotrados y los PC?
5. ¿Qué es un sistema operativo multitarea?
6. Si dispone de un PC, indique algunas situaciones en las que puede aprovecharse de sus capacidades multitarea.
7. Basándose en algún sistema de computadora con el que esté familiarizado, identifique dos paquetes de software de aplicación y dos paquetes de software de utilidad. A continuación explique por qué los ha clasificado de la forma que lo ha hecho.
8. a. ¿Cuál es el papel de la interfaz de usuario en un sistema operativo?
   b. ¿Cuál es el papel del núcleo de un sistema operativo?
9. ¿Qué estructura de directorios describe la ruta A/B/One.txt?
10. ¿Para qué se utiliza una tabla de procesos en un sistema operativo?
11. ¿Cuál es la función del planificador en un sistema operativo multitarea?
12. Enumere las ventajas de los sistemas operativos multitarea.
13. ¿Por qué es necesaria la memoria virtual?
14. Suponga que una computadora tiene 512 MB de memoria principal y que un sistema operativo necesita crear una memoria virtual del doble de tamaño utilizando páginas de 2 KB. ¿Cuántas páginas harán falta?
15. ¿Qué complicaciones pueden surgir en un sistema de tiempo compartido/multitarea si dos procesos requieren acceder al mismo archivo simultáneamente? ¿Existen casos en los que el administrador de archivos debería aprobar dichas solicitudes? ¿Existen casos en los que el administrador de archivos debería rechazar dichas solicitudes?
16. ¿Cuál es la diferencia entre software de aplicación y software del sistema? Proporcione un ejemplo de cada uno de ellos.
17. ¿Qué es un despachador? Explique las funciones de un despachador.
18. ¿Para qué se utiliza el BIOS?
19. ¿Qué es un semáforo?
20. Si dispone de un PC, anote la secuencia de actividades que puede observar al encenderlo. A continuación, determine qué mensajes aparecen en la pantalla de la computadora antes de que comience el propio proceso de arranque. ¿Qué software escribe estos mensajes?
21. Suponga que un sistema operativo de multiprogramación asigna franjas temporales de diez milisegundos y que la máquina ejecuta un promedio de cinco instrucciones por nanosegundo. ¿Cuántas instrucciones pueden ejecutarse en una única franja temporal?
22. Si una mecanógrafa escribe sesenta palabras por minuto (considerando que cada palabra tiene cinco caracteres), ¿cuánto tiempo pasaría entre la escritura de un carácter y el siguiente? Si un sistema de multiprogramación asignara franjas temporales en unidades de diez milisegundos e ignoramos el tiempo requerido para la conmutación de procesos, ¿cuántas franjas temporales podrían asignarse entre la escritura de dos caracteres consecutivos?
23. Suponga que un sistema operativo de multiprogramación asigna franjas temporales de 50 milisegundos. Si normalmente se tardan ocho milisegundos en colocar el cabezal de lectura/escritura del disco sobre la pista deseada y se requieren otros 17 milisegundos para que los datos deseados pasen bajo el cabezal de lectura/escritura, ¿qué porcentaje de la franja temporal de un programa podría invertirse esperando a que se complete una operación de lectura del disco? Si la máquina es capaz de ejecutar diez instrucciones por nanosegundo, ¿cuán-

tas instrucciones pueden ejecutarse durante ese periodo de espera? (Esta es la razón por la que cuando un proceso realiza una operación con un dispositivo periférico, los sistemas de multiprogramación finalizan la franja temporal de ese proceso y permiten a otro proceso ejecutarse, mientras que el primer proceso está esperando los servicios del dispositivo periférico.)

24. ¿Qué es un interbloqueo?

25. Decimos que un proceso está limitado por E/S si requiere muchas operaciones de E/S, mientras que un proceso que principalmente consta de cálculos dentro del sistema procesador/memoria se dice que está limitado por procesamiento. Si hay dos procesos esperando a que se les asigne una franja temporal, uno de ellos limitado por procesamiento y el otro limitado por E/S, ¿a cuál habría que dar prioridad? ¿Por qué?

26. ¿Cómo se conseguiría una mayor tasa de procesamiento en un sistema que estuviera ejecutando dos procesos en un entorno de multiprogramación: si ambos procesos son limitados por E/S (consulte el Problema 25) o si uno está limitado por E/S y el otro está limitado por procesamiento? ¿Por qué?

27. Escriba un conjunto de instrucciones que le indiquen al despachador del sistema operativo lo que hacer cuando se ha terminado la franja temporal de un proceso.

28. ¿Cómo se utiliza el *spooling* en un sistema operativo?

29. Identifique una situación en un sistema de multiprogramación en la que un proceso no consuma toda la franja temporal que se le ha asignado.

30. ¿Qué hace el procesador cuando se produce una interrupción?

31. Responda a cada una de las siguientes cuestiones en función del sistema operativo que normalmente utilice:
   a. ¿Cómo solicita al sistema operativo que copie un archivo de una ubicación a otra?
   b. ¿Cómo solicita al sistema operativo que muestre el directorio de un disco?
   c. ¿Cómo solicita al sistema operativo que ejecute un programa?

32. Responda a cada una de las siguientes cuestiones en función del sistema operativo que normalmente utilice:
   a. ¿Cómo restringe el sistema operativo el acceso, de forma que solo puedan utilizar la máquina los usuarios autorizados?
   b. ¿Cómo solicita al sistema operativo que muestre los procesos actualmente contenidos en la tabla de procesos?
   c. ¿Cómo indica al sistema operativo que no quiere que otros usuarios de la máquina tengan acceso a sus archivos?

***33.** Explique un uso importante de la instrucción de comprobación y configuración incluida en muchos lenguajes máquina. ¿Por qué es importante que todo el proceso de comprobación y configuración se implemente como una única instrucción?

***34.** Un banquero que solo tiene 100.000$ presta 50.000$ a un cliente y otros 50.000$ a otro. Posteriormente, ambos clientes vuelven con el cuento de que antes de poder pagar su deuda deben pedir prestados otros 10.000$ para completar los tratos comerciales en los que han invertido los préstamos anteriores. El banquero resuelve este interbloqueo pidiendo prestados los fondos adicionales a otra fuente y prestando ese dinero (con un incremento en la tasa de interés) a los dos clientes. ¿Cuál de las tres condiciones de interbloqueo ha eliminado el banquero?

***35.** Los estudiantes que quieren matricularse en la asignatura Modelismo ferroviario II en la universidad local tienen que conseguir permiso del profesor y pagar las tasas de laboratorio. Los dos requisitos se pueden satisfacer independientemente, en cualquier orden y en diferentes ubicaciones del campus. La matrícula está limitada a veinte alumnos y este límite es respetado tanto por el profesor, que solo concederá permiso a veinte estudiantes, como por el cajero de la univer-

sidad, que solo permitirá que veinte estudiantes paguen las tasas del laboratorio. Suponga que este sistema de registro ha dado como resultado que 19 estudiantes se han registrado para el curso, estando la plaza final reclamada por dos estudiantes, uno que solo ha obtenido permiso del profesor y otro que solo ha pagado las tasas del laboratorio. ¿Qué requisito para la aparición de interbloqueos se elimina mediante cada una de las siguientes soluciones al problema?

a. Se permite a ambos estudiantes asistir al curso.
b. Se reduce el tamaño de la clase a diecinueve, de modo que ninguno de los dos estudiantes obtiene permiso para asistir al curso.
c. Se deniega la entrada al curso a ambos estudiantes y se le proporciona la vigésima plaza a un tercer estudiante.
d. Se decide que el único requisito para matricularse en el curso es el pago de las tasas. Por tanto, el estudiante que ha pagado las tasas puede asistir al curso, mientras que al otro se le deniega la asistencia.

***36.** Puesto que cada área de la pantalla de una computadora solo puede ser utilizada por un proceso cada vez (en caso contrario, la imagen en la pantalla sería ilegible), dichas áreas son recursos no compartibles asignados por el administrador de ventanas. ¿Cuál de las tres condiciones necesarias para la aparición de interbloqueos elimina el administrador de ventanas para impedir que se produzcan interbloqueos?

***37.** Suponga que clasificamos cada recurso no compartible de una computadora en tres niveles: recursos de nivel 1, de nivel 2 y de nivel 3. Suponga además que a cada proceso del sistema le exigimos que solicite los recursos que necesita de acuerdo con dicha clasificación; es decir, debe solicitar todos los recursos de nivel 1 requeridos antes de solicitar ningún recurso del nivel 2. Una vez recibidos los recursos de nivel 1, puede solicitar todos los recursos necesarios de nivel 2, y así sucesivamente. ¿Puede producirse un interbloqueo en este tipo de sistema? ¿Por qué?

***38.** Disponemos de dos brazos de robot programados para elevar aparatos de una cinta transportadora, comprobar que cumplen las tolerancias y colocarlos en uno de dos contenedores dependiendo de los resultados de las pruebas. Los aparatos llegan de uno en uno con un intervalo suficiente de tiempo entre ellos. Para impedir que ambos brazos traten de elevar el mismo aparato, las computadoras que controlan dichos brazos de robot comparten una celda de memoria común. Si un brazo está disponible cuando un aparato se aproxima, la computadora que lo controla lee el valor de la celda común. Si el valor es distinto de cero, el brazo deja que el aparato pase, en caso contrario, la computadora de control coloca un valor distinto de cero en la celda de memoria, ordena al brazo de robot que levante el aparato y vuelve a escribir el valor cero en la celda de memoria después de que dicha acción se ha completado. ¿Qué secuencia de sucesos podría conducir a que los dos brazos de robot intentaran levantar el mismo aparato?

39.** Identifique el uso de una cola en el proceso de envío de datos de salida hacia una impresora mediante ***spooling.

***40.** Decimos que un proceso que está esperando a que se le asigne una franja temporal sufre de **inanición** cuando a ese proceso nunca se le asigna una franja temporal.

a. El pavimento situado en el centro de una intersección puede considerarse como un recurso no compartido, por el que compiten los vehículos que se aproximan a esa intersección. Se utiliza un semáforo, en lugar de un sistema operativo, para controlar la asignación de este recurso. Si el semáforo es capaz de detectar la cantidad de tráfico que llega desde cada dirección y está programado para dar luz verde a la dirección en la que hay un tráfico más intenso, la otra dirección en la que el tráfico es menos intenso podría sufrir de inanición. ¿Cómo se evitaría la inanición?

b. ¿En qué sentido puede un proceso sufrir de inanición, si el despachador asigna siempre las franjas temporales de acuerdo con un sistema de prioridades, en el que la prioridad de cada proceso permanece fija. (***Sugerencia:*** ¿cuál es la prioridad del proceso que acaba de completar su franja temporal, comparada con la de los procesos que están esperando y, en consecuencia, qué rutina obtendrá la siguiente franja temporal?) ¿Cómo cree que evitan este problema numerosos sistemas operativos?

***41.** ¿En qué se parecen el interbloqueo y la inanición? (Consulte el Problema 40.) ¿Cuál es la diferencia entre interbloqueo e inanición?

***42.** El problema siguiente se conoce como el problema de "la cena de los filósofos", que originalmente fue propuesto por E. W. Dijkstra y actualmente forma parte del folklore de las Ciencias de la computación.

Imagine que hay cinco filósofos sentados alrededor de una mesa redonda. Delante de ellos hay un plato de espagueti. Hay cinco tenedores en la mesa, situados entre cada dos platos. Cada filósofo quiere alternar entre las actividades de pensar y comer. Para comer, un filósofo necesita estar en posesión de los dos tenedores adyacentes a su plato.

Identifique las posibilidades de interbloqueo e inanición (consulte el Problema 40) presentes en el problema de la cena de los filósofos.

***43.** ¿Qué problema surge si se van acortando cada vez más las longitudes de las franjas temporales en un sistema de multiprogramación? ¿Y qué sucede si se van alargando cada vez más?

***44.** A medida que se han ido desarrollando las Ciencias de la computación, los lenguajes máquina se han ampliado para proporcionar instrucciones especializadas. En la Sección 3.4 hemos comentado tres de esas instrucciones de lenguaje máquina utilizadas ampliamente por los sistema operativos. ¿ Cuáles son esas instrucciones?

45. Identifique dos actividades que puedan ser realizadas por el administrador de un sistema operativo, pero no por un usuario típico.

46. ¿Cómo impide un sistema operativo que un proceso acceda al espacio de memoria de otro proceso?

47. Suponga que una contraseña está formada por una cadena de ocho caracteres del alfabeto inglés (compuesto por 26 caracteres), los números decimales (10 caracteres) y dos símbolos especiales ($, _). Si pudiéramos comprobar cada posible contraseña en un milisegundo, ¿cuánto tardaríamos en probar todas las posibles contraseñas?

48. ¿Por qué los procesadores diseñados para sistemas operativos multitarea son capaces de operar en distintos niveles de privilegio?

49. Explique la función del gestor de memoria desempeñada por el núcleo de un sistema operativo.

50. Identifique tres formas en las que un proceso podría tratar de vencer la seguridad de una computadora, si el sistema operativo no lo impidiera.

51. ¿Cuáles son las ventajas de los sistemas operativos multinúcleo respecto a los sistemas de un solo núcleo?

52. ¿Para qué se utiliza el programa cargador de arranque ?

53. ¿Quién es el responsable del "estilo" de la interfaz gráfica de usuario de un sistema operativo?

54. ¿Es Internet Explorer una parte del sistema operativo Microsoft Windows?

55. ¿Qué es un teléfono inteligente?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Suponga que está utilizando un sistema operativo multiusuario que le permite ver el nombre de los archivos pertenecientes a otros usuarios, así como el contenido de aquellos archivos que no estén protegidos de alguna manera. ¿Cree que el visualizar esa información sin permiso es similar a moverse por el domicilio de alguien sin permiso, o se parece más a leer las revistas depositadas en una sala común, como por ejemplo en la sala de espera de un médico?
2. Cuando disponemos de acceso a una computadora multiusuario, ¿qué responsabilidades tenemos a la hora de seleccionar nuestra contraseña?
3. Si un fallo en la seguridad de un sistema operativo permite a un programador malicioso obtener acceso no autorizado a datos confidenciales, ¿hasta qué punto cabe atribuir la responsabilidad al desarollador del sistema operativo?
4. ¿Es responsabilidad nuestra cerrar nuestra vivienda para que los intrusos no puedan entrar, o es responsabilidad del resto de las personas no entrar en nuestra vivienda sin ser invitados? ¿Es responsabilidad de un sistema operativo proteger el acceso a una computadora y a su contenido, o es responsabilidad de los piratas informáticos (***hackers***) no introducirse en la máquina?
5. En ***Walden,*** Henry David Thoreau argumenta que nos hemos convertido en herramientas de nuestras herramientas; es decir, en lugar de obtener beneficios de las herramientas de las que disponemos, invertimos nuestro tiempo en obtener y mantener esas herramientas. ¿Hasta qué punto es esto cierto en lo que respecta a la computación? Si tenemos una computadora personal, ¿cuánto tiempo invertimos en ganar el dinero necesario para adquirirla, en aprender a usar su sistema operativo, en aprender a utilizar su software de aplicación y de utilidad, en mantenerla y en descargar actualizaciones para su software en comparación con la cantidad de tiempo que pasamos obteniendo beneficios de ella? Cuándo la utilizamos, ¿se trata de un tiempo bien invertido? ¿Cómo es usted más activo socialmente, con una computadora o sin ella?

Lecturas adicionales

Bishop, M. ***Introduction to Computer Security.*** Boston, MA: Addison-Wesley, 2005.

Davis, W. S. y T. M. Rajkumar. ***Operating Systems***: ***A Systematic View***, 6ª ed. Boston, MA: Addison-Wesley, 2005.

Deitel, H. M., P. J. Deitel y D. R. Choffnes. *Operating Systems*, 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2005.

Nutt, G. *Operating Systems: A Modern Approach*, 3ª ed. Boston, MA: Addison-Wesley, 2004.

Rosenoer, J. *CyberLaw, The Law of the Internet*. Nueva York: Springer, 1997.

Silberschatz, A., P. B. Galvin y G. Gagne. *Operating System Concepts*, 8ª ed., Nueva York: Wiley, 2008.

Stallings, W. *Operating Systems*, 5ª ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Tanenbaum, A. S. *Modern Operating Systems*, 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2008.

CAPÍTULO

4

Redes e Internet

En este capítulo vamos a ocuparnos de ese campo de las Ciencias de la computación que describimos mediante el término redes y que abarca el estudio de la forma en que pueden conectarse unas computadoras con otras para compartir información y recursos. Nuestro estudio incluirá la construcción y el funcionamiento de redes, las aplicaciones de red y las cuestiones de seguridad. Uno de los temas más importantes será una red de redes concreta de alcance mundial a la que denominamos Internet.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

La necesidad de compartir información y recursos entre diferentes computadoras ha llevado a la aparición de sistemas de computación conectados denominados **redes**, en los que las computadoras están conectadas de manera que los datos pueden transferirse de una máquina a otra. En estas redes, los usuarios de las computadoras pueden intercambiar mensajes y compartir recursos tales como capacidades de impresión, paquetes software y dispositivos de almacenamiento de datos, que están dispersos a través de todo el sistema. El software subyacente necesario para dar soporte a tales aplicaciones ha ido creciendo desde los simples paquetes de utilidades hasta convertirse en un sistema en expansión de software de red que proporciona una sofisticada infraestructura distribuida. En cierto sentido, el software de red está evolucionando hacia un sistema operativo de red. En este capítulo vamos a estudiar este campo en expansión de las Ciencias de la computación.

4.1 Fundamentos de las redes

Comenzaremos nuestro estudio acerca de las redes presentando diversos conceptos básicos de este campo.

Clasificación de las redes

Las redes de computadoras suele clasificarse como **redes de área local** (LAN, *Local Area Network*), **redes de área metropolitana** (MAN, *Metropolitan Area Network*) o **redes de área extensa** (WAN, *Wide Area Network*). Normalmente, una LAN consta de una colección de computadoras situadas en un mismo edificio o en un complejo de edificios. Por ejemplo, las computadoras de un campus universitario o las de una fábrica pueden estar conectadas mediante una LAN. Una MAN es una red de tamaño intermedio, como por ejemplo una que abarque todo un pueblo de pequeño tamaño. Una red WAN conecta máquinas situadas a grandes distancias, quizá en ciudades próximas o en lados opuestos del mundo.

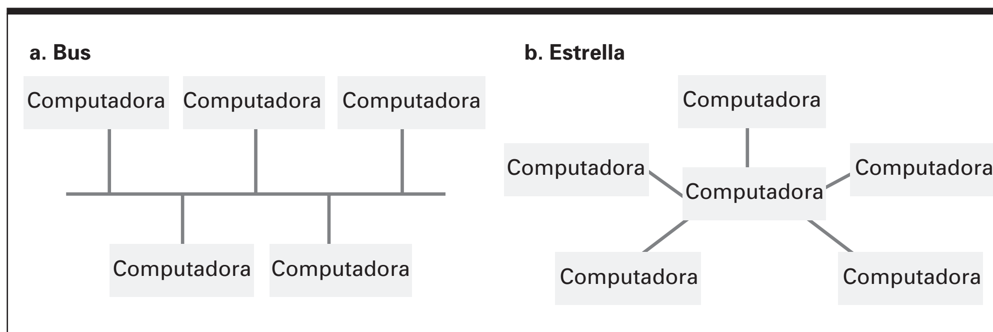
Otro método de clasificación de las redes se fija en si la operación interna de la red está basada en diseños que son de dominio público o en innovaciones que están controladas y son propiedad de una entidad concreta, como por ejemplo una persona o una corporación. Una red del primer tipo se dice que es una red **abierta**; una red del segundo tipo se denomina red **cerrada** o, en ocasiones, red **propietaria**. Los diseños de red abierta pueden circular libremente y a menudo su popularidad crece hasta el punto de que terminan prevaleciendo sobre los enfoques propietarios, cuyas aplicaciones están restringidas por los acuerdos de licencia y las condiciones fijadas en los correspondientes contratos.

Internet (una red de redes mundial muy popular que estudiaremos en este capítulo) es un sistema abierto. En particular, la comunicación a través de Internet está gobernada por un conjunto abierto de estándares conocido con el nombre de conjunto de protocolos TCP/IP, que es el tema del que nos ocuparemos en la Sección 4.4. Cualquier persona es libre de utilizar esos estándares sin pagar licencias ni firmar ningún tipo de acuerdo. Por el contrario, una empresa como Novell Inc. puede desarrollar sistemas propietarios para los que decide mantener sus derechos de propiedad, lo que permite a la empresa obtener ingresos de la venta o alquiler de los correspondientes productos.

Otra tercera forma de clasificar las redes se basa en la topología de la red, donde el término topología hace referencia al patrón con el que las máquinas están conectadas. Dos de las topologías más populares son la de bus, en la que todas las máquinas están conectadas a una línea común de comunicaciones denominada bus (Figura 4.1a) y la topología en estrella, en la que una máquina sirve como punto central al que todas las demás se conectan (Figura 4.1b). La topología de bus fue popularizada en la década de 1990, cuando se la implementó mediante una serie de estándares conocidos con el nombre de Ethernet y las redes Ethernet continúan siendo una de los sistemas de red más populares que actualmente se utilizan. El origen de la topología en estrella se remonta a la década de 1970. Evolucionó a partir del paradigma de una computadora central de gran tamaño que presta servicio a múltiples usuarios. A medida que los terminales simples utilizados por dichos usuarios crecieron hasta convertirse ellos mismos en pequeñas computadoras, fueron emergiendo las redes en estrella. Hoy día, la configuración en estrella es popular en las redes inalámbricas, en las que la comunicación se lleva a cabo por medio de emisiones de radio y en los que la máquina central, denominada **punto de acceso** (AP, *Access Point*), sirve como punto focal en torno al cual se coordina toda la comunicación.

La diferencia entre una red en bus y otra en estrella no siempre resulta obvia examinando la disposición física de los equipos. La distinción radica en si las máquinas de la red se ven a sí mismas comunicándose directamente entre ellas a través del bus común, o indirectamente a través de una máquina central intermedia. Por ejemplo, una red en bus puede no tener el aspecto de un bus de gran tamaño al que las computadoras se conectan mediante cortos enlaces como se muestra en la Figura 4.1. En lugar de ello, podría tener un bus muy corto con largos enlaces hasta la máquinas individuales, lo que implica que la red podría tener un aspecto más similar al de una red en estrella. De hecho, en ocasiones, las redes en bus se crean tendiendo enlaces desde cada computadora a una ubicación central, donde los distintos enlaces se conectan a un dispositivo denominado **concentrador** (*hub*). Este concentrador, en realidad, no es más que una especie de bus muy corto. Todo lo que hace es reemitir cualquier señal que recibe, quizá con algo de amplificación (hacia todas las máquinas que están conectadas a él). El resultado es una red que parece tener una topología en estrella, pero que opera como una red en bus.

Figura 4.1 Dos topologías de red populares.



Protocolos

Para que una red funcione de forma fiable es importante establecer reglas a las que se ajusten todas las actividades. Tales reglas se denominan **protocolos**. Desarrollando y adoptando estándares de protocolo, los fabricantes son capaces de construir productos para aplicaciones de red que sean compatibles con productos de otros fabricantes. Por ello, el desarrollo de estándares de protocolo es un proceso indispensable para el avance de las tecnologías de red.

Como introducción al concepto de protocolo, consideremos el problema de la coordinación de la transmisión de mensajes entre las computadoras de una red. Sin reglas que gobiernen esta comunicación, todas las computadoras podrían insistir en transmitir los mensajes al mismo tiempo o podrían no proporcionar ayuda a otras máquinas cuando estas las requirieran.

En una red en bus basada en los estándares Ethernet, el derecho de transmitir mensajes está controlado por el protocolo **CSMA/CD** (*Carrier Sense Multiple Access with Collision Detection*, Acceso múltiple por detección de portadora con detección de colisiones). Este protocolo dicta que todos los mensajes sean difundidos a todas las máquinas del bus (Figura 4.2). Cada máquina monitoriza todos los mensajes, pero solo se queda con aquellos que están dirigidos a ella. Para transmitir un mensaje, una máquina espera hasta que el bus esté en silencio, en cuyo momento comienza a transmitir mientras continúa monitorizando el bus. Si otra máquina comienza a transmitir también al mismo tiempo, las dos detectarán la colisión y harán una pausa durante un corto periodo de tiempo, de longitud aleatoria, antes de tratar de transmitir de nuevo. El resultado es un sistema similar al que un grupo de personas utilizaría en una conversación. Si dos personas comienzan a hablar al mismo tiempo, ambas se paran. La diferencia es que esas personas podrían enzarzarse en una serie de frases tales como, "Lo siento, ¿qué ibas a decir?", "No, no. Tu primero", mientras que con el protocolo CSMA/CD cada máquina simplemente vuelve a intentarlo más adelante.

Observe que CSMA/CD no es compatible con las redes inalámbricas en estrella, en la que todas las máquinas se comunican a través de un punto de acceso (AP) central. La razón es que una máquina puede no ser capaz de detectar que sus transmisiones están colisionando con las de otra. Por ejemplo, la máquina puede no escuchar a la otra porque su propia señal se superpone a la de la otra máquina. Otra causa podría ser que las señales de las distintas máquinas estén bloqueadas unas con respecto a otras debido a los objetos o la distan-

Figura 4.2 Comunicación a través de una red en bus.

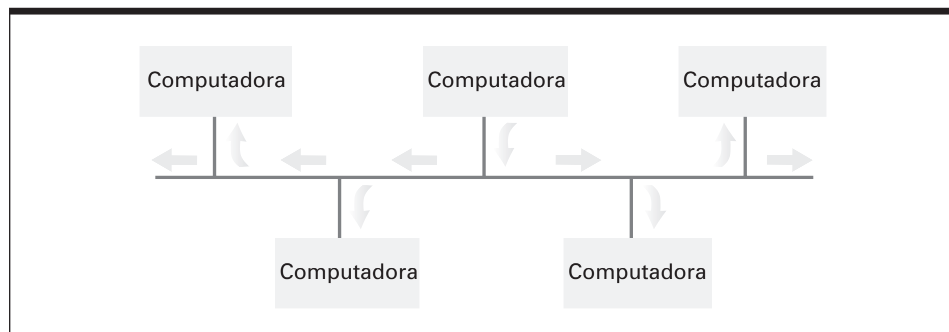
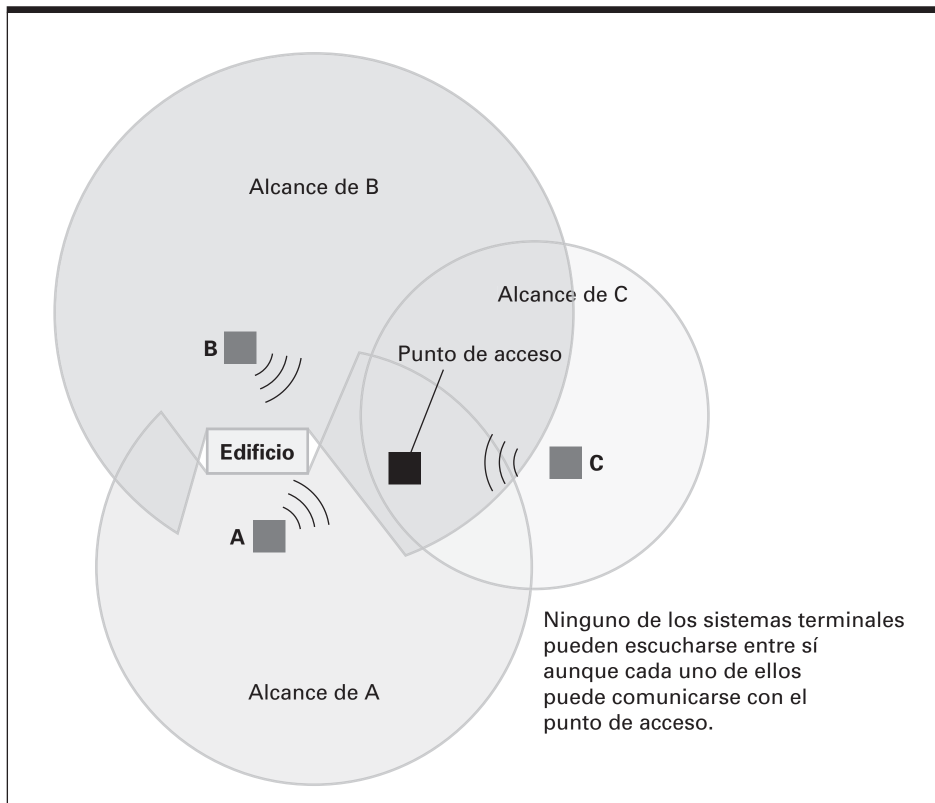


Figura 4.3 El problema del terminal oculto.



cia existente, aun cuando todas las máquinas puedan comunicarse con el AP central (una condición que se conoce con el nombre de **problema del terminal oculto**, Figura 4.3). El resultado es que las redes inalámbricas adoptan la política de tratar de *evitar* las colisiones en lugar de tratar de *detectarlas*. Ese tipo de políticas se denominan **CSMA/CA** (*Carrier Sense Multiple Access with Collision Avoidance*, Acceso múltiple por detección de portadora con evitación de colisiones), muchas de las cuales están estandarizados por IEEE (consulte el recuadro "IEEE, Instituto de Ingenieros Eléctricos y Electrónicos", en el Capítulo 7) dentro de los protocolos definidos en la norma IEEE 802.11 y a los que comúnmente se les llama **WiFi**. Hay que recalcar que los protocolos de evitación de colisiones están diseñados para evitar las colisiones y pueden no eliminarlas completamente. En aquellos casos en que las colisiones llegan a producirse se hace necesario retransmitir los mensajes.

La técnica más común para la evitación de colisiones se basa en proporcionar ventaja a aquellas máquinas que ya han estado esperando a tener una oportunidad de transmitir. El protocolo utilizado es similar al CSMA/CD de Ethernet. La diferencia básica es que, cuando una máquina necesita por primera vez transmitir un mensaje y se encuentra con que el canal de comunicación está en silencio, no comienza a transmitir directamente. En lugar de ello, espera durante un periodo corto de tiempo y luego empieza a transmitir solo si el canal ha permanecido en silencio a lo largo de dicho periodo. Si se detecta un canal

Ethernet

Ethernet es un conjunto de estándares que permiten implementar una red LAN con una topología en bus. Su nombre viene de su diseño Ethernet original en el que las máquinas se conectaban mediante un cable coaxial al que llamaban éter (ether). Originalmente desarrollado en la década de 1930 y ahora estandarizado por el IEEE como parte de la familia de estándares IEEE 802, Ethernet es uno de los métodos más comunes utilizados para conectar máquinas PC en red. De hecho, las controladoras Ethernet se han convertido en un componente estándar de los PC actualmente disponibles en el mercado de consumo.

Hoy día, existen en realidad varias versiones de Ethernet que reflejan los avances experimentados en la tecnología y la disponibilidad de tasas de transferencia más altas. Todos esos estándares comparten, sin embargo, una serie de características comunes que son distintivas de la familia Ethernet. Entre ellas están el formato con el que se empaquetan los datos para su transmisión, el uso de la codificación Manchester (un método de representar ceros y unos en el que un cero se representa mediante una señal descendente y un uno mediante una señal ascendente) para la transmisión de los bits y la utilización de CSMA/CD para el control del derecho a transmitir.

ocupado durante este proceso, la máquina espera durante un periodo de tiempo aleatorio antes de intentarlo de nuevo. Una vez transcurrido este periodo, se permite a la máquina apropiarse del canal silencioso sin ningún tipo de espera adicional. Esto significa que se evitan las colisiones entre los "recién llegados" y aquellos que ya han estado esperando, porque a un "recién llegado" no se le permite reclamar la utilización del canal silencioso hasta haber dado la oportunidad de comenzar a las otras máquinas que puedan haber estado esperando.

Sin embargo, este protocolo no resuelve el problema del terminal oculto. Después de todo, cualquier protocolo basado en distinguir entre un canal silencioso y otro ocupado requiere que cada estación individual sea capaz de escuchar a todas las demás. Para solucionar este problema, algunas redes WiFi requieren que cada máquina envíe un mensaje de "solicitud" corto al punto de acceso y espera hasta que el punto de acceso confirme dicha solicitud antes de transmitir el mensaje completo. Si el punto de acceso está ocupado porque está comunicándose con un terminal oculto, ignorará la solicitud y la máquina solicitante sabrá que tiene que esperar. En caso contrario, el punto de acceso confirmará la solicitud y la máquina sabrá que puede transmitir sin ningún problema. Observe que todas las máquinas de la red podrán escuchar las confirmaciones enviadas desde el punto de acceso y, por tanto, tendrán una idea correcta de si el punto de acceso está ocupado en un momento determinado, aún cuando ellas mismas no puedan escuchar las transmisiones que están teniendo lugar.

Combinación de redes

En ocasiones, es necesario conectar redes existentes para formar un sistema de comunicaciones más amplio. Esto se puede hacer conectando las redes para formar una versión de mayor tamaño del mismo "tipo" de red. Por ejemplo, en el caso de redes en bus basadas en los protocolos Ethernet, a menudo es posible

conectar los buses para formar un único bus de gran longitud. Esto se lleva a cabo por medio de diferentes dispositivos que se conocen con los nombres de repetidores, puentes y conmutadores, siendo las diferencias entre unos y otros bastante sutiles pero significativas. El más simple de estos dispositivos es el **repetidor**, que no es otra cosa que un dispositivo que simplemente pasa las señales de un lado a otro entre los dos buses originales (usualmente con algún tipo de amplificación) sin considerar el significado de las señales (Figura 4.4a).

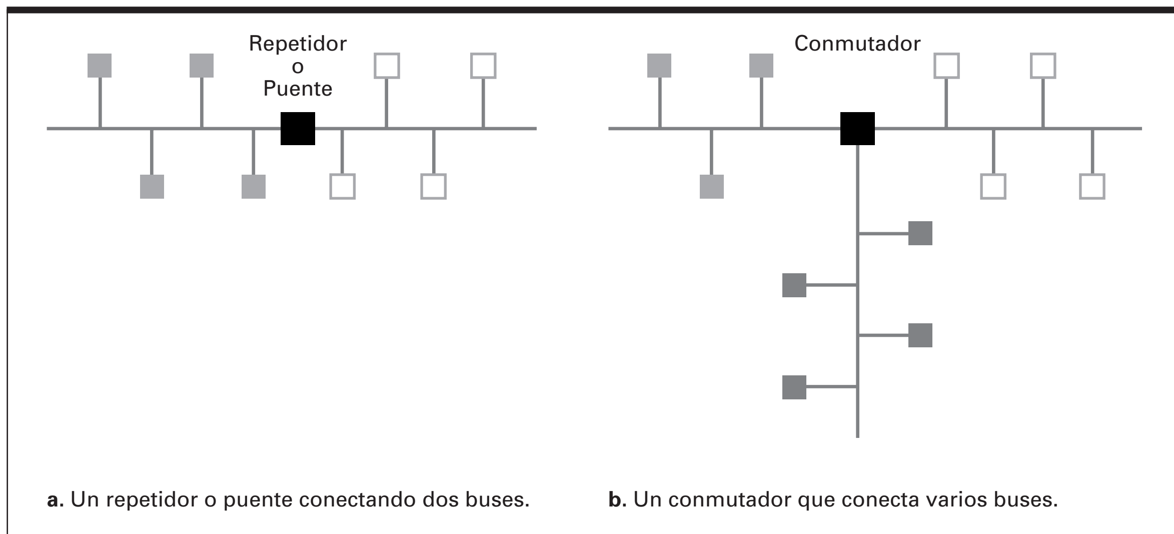
Un **puente** es similar a un repetidor, pero más complejo. Al igual que el repetidor, permite conectar dos buses, pero el puente no pasa necesariamente todos los mensajes a través de la conexión. En su lugar, examina la dirección de destino que acompaña a cada mensaje y reenvía un mensaje a través de la conexión solo cuando está destinado a una computadora situada al otro lado. De este modo, dos máquinas que residan en el mismo lado de un puente pueden intercambiar mensajes sin interferir en las comunicaciones que están teniendo lugar al otro lado. Un puente permite conseguir un sistema más eficiente que un repetidor.

Un **conmutador** es básicamente un puente con múltiples conexiones, lo que le permite conectar varios buses en lugar de solo dos. De ese modo, un conmutador permite construir una red compuesta por varios buses que se extienden a partir del conmutador como los radios de una rueda (Figura 4.4b). Como en el caso de un puente, el conmutador analiza las direcciones de destino de todos los mensajes y solo reenvía aquellos mensajes que están destinados a otros radios de esa rueda. Además, cada mensaje solo se retransmite hacia el radio apropiado, minimizando así el tráfico en cada uno de los radios.

Es importante observar que cuando las redes están conectadas mediante repetidores, puentes y conmutadores, el resultado es una única red de mayor tamaño. Todo el sistema opera de la misma forma (utilizando los mismos protocolos) que cada una de las redes originales de menor tamaño.

Sin embargo, en ocasiones, las redes que hay que conectar tienen características incompatibles. Por ejemplo, las características de una red WiFi no se

Figura 4.4 Construcción de una red en bus de gran tamaño a partir de otras más pequeñas.



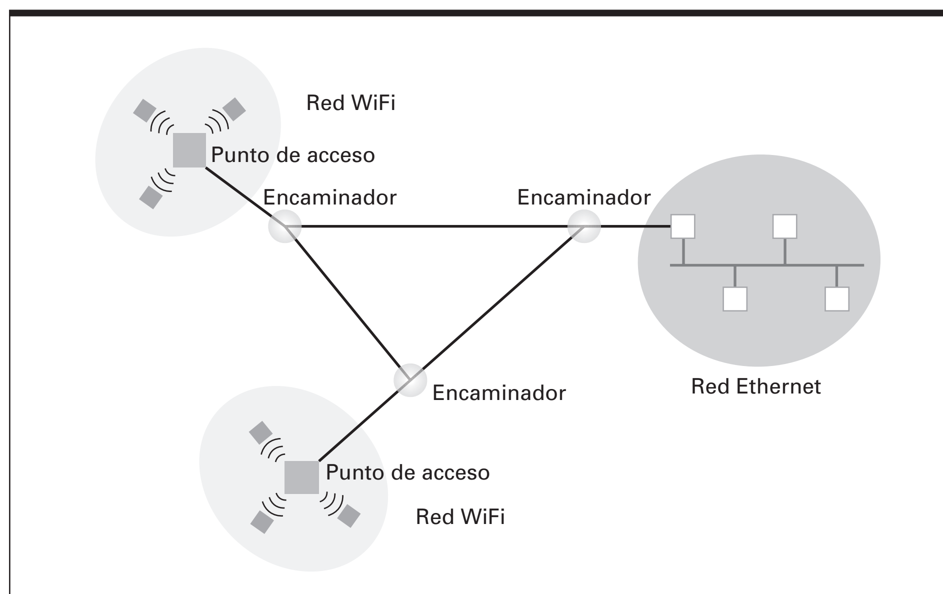
a. Un repetidor o puente conectando dos buses. **b.** Un conmutador que conecta varios buses.

pueden compatibilizar fácilmente con las de una red Ethernet. En estos casos, las redes deben conectarse de forma que se construya una red de redes, lo que se conoce como una **interred**, en la que las redes originales mantienen su individualidad y continúan funcionando como redes autónomas. (La traducción del término interred al inglés es ***internet***, pero es necesario distinguir este término de ***Internet***. La red Internet, escrita con mayúscula, ***I***, hace referencia a una interred concreta de alcance mundial que estudiaremos en posteriores secciones de este capítulo. Existen otros muchos ejemplos de interredes. De hecho, la comunicación telefónica tradicional se gestionaba mediante sistemas de interredes mundiales mucho antes de que Internet se hiciera popular.)

La conexión entre redes para formar una interred se gestiona mediante una serie de dispositivos conocidos como **encaminadores** (*router*), que son computadoras de uso especial utilizadas para el reenvío de mensajes. Observe que la tarea de un encaminador es diferente de la de los repetidores, puentes y conmutadores, en el sentido de que los encaminadores proporcionan enlaces entre redes, mientras permiten a cada una de esas redes mantener sus características internas distintivas. Por ejemplo, en la Figura 4.5 se muestran dos redes WiFi en estrella y una red Ethernet con topología en bus conectadas mediante encaminadores. Cuando una máquina en una de las redes WiFi quiere enviar un mensaje a una máquina de la red Ethernet, primero envía el mensaje al punto de acceso de su red. A partir de ahí, el punto de acceso envía el mensaje a su encaminador asociado y este encaminador reenvía el mensaje al encaminador de la red Ethernet. Allí, el mensaje se entrega a una máquina conectada al bus y dicha máquina reenvía entonces el mensaje hasta su destino final dentro de la red Ethernet.

La razón de que a los encaminadores se los llame así es que su propósito consiste en reenviar los mensajes a las direcciones apropiadas. Este proceso de reen-

Figura 4.5 Encaminadores que conectan dos redes WiFi y una red Ethernet para formar una interred.



vío se basa en un sistema de direccionamiento global de la interred, en el que a todos los dispositivos de la interred (incluyendo las máquinas de las redes originales y los propios encaminadores) se les asignan direcciones distintivas (por tanto, cada máquina de las redes originales dispondrá de dos direcciones: su dirección original "local" dentro de su propia red y su dirección dentro de la interred). Una máquina que quiera enviar un mensaje a otra máquina situada en una red distante asociará al mensaje la dirección del destino dentro de la interred y dirigirá dicho mensaje hacia su encaminador local. A partir de ahí, el mensaje será reenviado en la dirección apropiada. Para esta tarea de reenvío, cada encaminador mantiene una **tabla de reenvío** que contiene todo el conocimiento que el encaminador posee acerca de la dirección en la que hay que enviar los mensajes, dependiendo de cuál sea su dirección de destino.

El "punto" en el que una red se conecta a una interred se suele denominar **pasarela** (*gateway*), porque sirve como lugar de paso entre la red y el mundo exterior. Hay pasarelas de muchos tipos y por tanto dicho término se emplea de forma más bien vaga. En muchos casos, la pasarela de una red es simplemente el encaminador mediante el cual se comunica con el resto de la interred. En otros casos, el término *pasarela* puede utilizarse para referirse a algo más que un simple encaminador. Por ejemplo, en la mayoría de las redes WiFi residenciales que están conectadas a Internet, el término *pasarela* se refiere conjuntamente tanto al punto de acceso de la red como al encaminador conectado a dicho punto de acceso, debido a que esos dos dispositivos suelen empaquetarse en una misma unidad.

Métodos de comunicación entre procesos

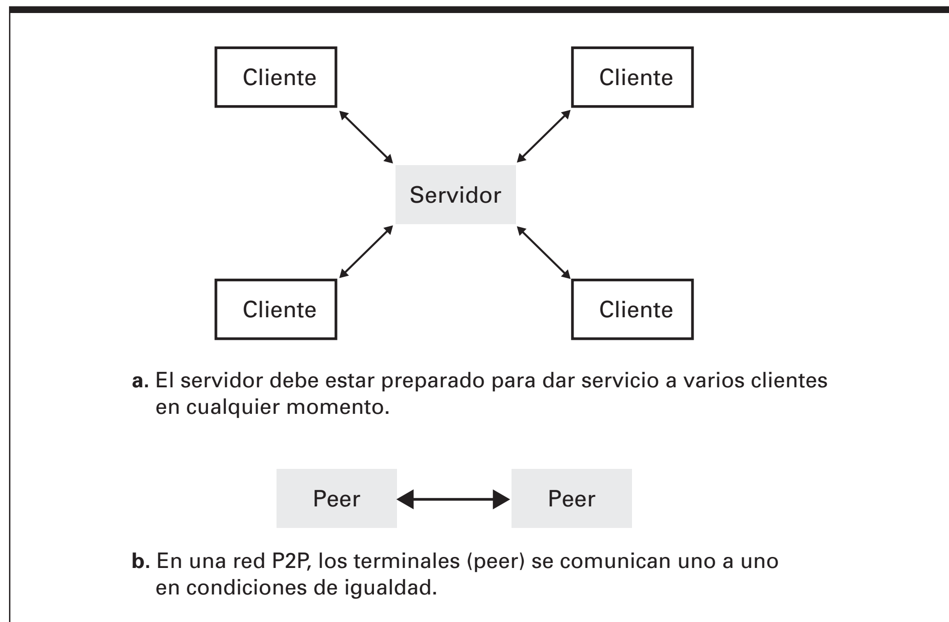
Las diversas actividades (o procesos) que se ejecutan en las distintas computadoras de una red (o incluso que se ejecutan en una misma máquina mediante los mecanismos de tiempo compartido/multitarea) deben a menudo comunicarse entre sí para coordinar sus acciones y realizar la tareas que tienen asignadas. Dicha comunicación entre procesos se denomina **comunicación interprocesos**.

Un convenio bastante popular que se utiliza para la comunicación interprocesos es el modelo **cliente/servidor**. Este modelo define los papeles básicos que desempeñan los procesos bien como **cliente**, que realiza solicitudes a otros procesos, o bien como **servidor**, que satisface las solicitudes realizadas por los clientes.

Una de las primeras aplicaciones del modelo cliente/servidor se desarrolló en las redes que conectaban todas las computadoras de oficinas. En ese tipo de escenario, se conectaba una única impresora de alta calidad a la red para que estuviera disponible para todas las máquinas de una oficina. En este caso, la impresora desempeñaba el papel de servidor (a menudo denominado **servidor de impresión**), y las demás máquinas se programaban para desempeñar el papel de clientes que enviaban solicitudes al servidor de impresión.

Otras de las primeras aplicaciones del modelo cliente/servidor se utilizaba para reducir el coste del almacenamiento en disco magnético, al mismo tiempo que se eliminaba la necesidad de mantener copias duplicadas de los registros. En ese escenario, se equipaba a una de las máquinas de la red con un sistema de almacenamiento masivo de alta capacidad (usualmente un disco magnético)

Figura 4.6 El modelo cliente/servidor comparado con el modelo P2P (peer-to-peer).



que contenía todos los registros de la organización. Las otras máquinas de la red solicitaban acceso a los registros a medida que lo necesitaban. De este modo, la máquina que contenía físicamente los registros hacía el papel del servidor (denominado **servidor de archivos**), mientras que las demás máquinas hacían el papel de clientes que solicitaban el acceso a los archivos almacenados en el servidor de archivos.

Hoy día, el modelo cliente/servidor se utiliza ampliamente en las aplicaciones de red, como veremos posteriormente en el capítulo. Sin embargo, el modelo cliente/servidor no es el único modelo posible de comunicación entre procesos. Otro modelo es el **P2P** (*peer-to-peer,* entre pares). Mientras que el modelo cliente/servidor implica que hay un proceso (el servidor) que proporciona servicio a numerosos otros procesos (los clientes), el modelo P2P implica que existen procesos que proporcionan servicio a otros y reciben servicio también de ellos (Figura 4.6). Además, mientras que un servidor debe ejecutarse de manera continua para estar preparado para dar servicio a los clientes en cualquier momento, el modelo P2P implica usualmente procesos que se ejecutan de manera temporal. Por ejemplo, entre las aplicaciones del modelo peer-to-peer se incluye la mensajería instantánea, mediante la que las personas pueden mantener una conversación escrita a través de Internet, así como aquellas otras situaciones en las que hay un cierto número de personas utilizando juegos interactivos de naturaleza competitiva.

El modelo peer-to-peer también es un método muy popular de distribuir archivos a través de Internet, como por ejemplo grabaciones musicales y películas. En este caso, un terminal puede recibir un archivo de otro y luego proporcionar dicho archivo a otros terminales. La colección de todos los terminales (*peers*) que participan en ese tipo de distribución se denomina en ocasiones enjambre (*swarm*). La técnica de distribución de archivos basada en un

enjambre de terminales contrasta con las técnicas anteriores que empleaban el modelo cliente/servidor, estableciendo un punto de distribución central (el servidor) desde el que los clientes descargaban los archivos (o donde al menos encontraban las fuentes para dichos archivos).

Una de las razones por las que el modelo P2P está sustituyendo al modelo cliente/servidor para la compartición de archivos es que dicho modelo distribuye la tarea de dar servicio entre muchos terminales iguales en lugar de concentrarla en un único servidor. Esta falta de una base de operaciones centralizada permite obtener un sistema más eficiente. Lamentablemente, otra razón para la popularidad de los sistemas de distribución de archivos basados en el modelo P2P es que, en una serie de casos de cuestionable legalidad, la falta de un servidor central hace que los esfuerzos legales por imponer las leyes de propiedad intelectual resulte más difícil. Sin embargo, existen numerosos casos en los que algunos individuos han descubierto que "difícil" no significa "imposible", habiéndose tenido que enfrentar a graves responsabilidades debido a la violación de los derechos de autor.

A menudo, tenemos la oportunidad de leer o escuchar el término ***red peer-to-peer***, que constituye un ejemplo de cómo puede llegar a evolucionar un uso inadecuado de la terminología cuando la comunidad no técnica adopta términos técnicos. El término ***peer-to-peer*** hace referencia a un sistema mediante el que dos sistemas se comunican a través de una red (o interred). No es una propiedad de la red (o de la interred). Un proceso podría utilizar el modelo peer-to-peer para comunicarse con otro proceso y luego posteriormente utilizar el modelo cliente/servidor para comunicarse con otro proceso distinto a través de la misma red. Por tanto, sería mucho más apropiado hablar de comunicación por medio del modelo peer-to-peer que de comunicación a través de una red peer-to-peer.

Sistemas distribuidos

Con el éxito de la tecnología de redes, la interacción entre computadoras a través de redes se ha convertido en algo común y que tiene múltiples facetas. Muchos sistemas software modernos como sistemas globales de extracción de información, los sistemas empresariales de contabilidad e inventario, los juegos de computadora e incluso el software que controla la propia infraestructura de una red se diseñan como **sistemas distribuidos**, lo que quiere decir que constan de unidades software que se ejecutan como procesos independientes en diferentes computadoras.

Los primeros sistemas distribuidos se desarrollaron de forma independiente partiendo de cero. Pero hoy día las investigaciones revelan que existe una infraestructura común a todos estos sistemas la cual incluye sistemas tales como los de comunicaciones y de seguridad. A su vez, se han hecho esfuerzos para construir sistemas prefabricados que proporcionen esta infraestructura básica y que permitan, por tanto, construir aplicaciones distribuidas desarrollando exclusivamente aquella parte del sistema que sea característica de la aplicación.

Hoy día, son varios los tipos de sistemas de computación distribuidos de uso común. La **computación en cluster** describe un sistema distribuido en el que múltiples computadoras independientes trabajan estrechamente de manera conjunta para proporcionar potencia de computación o servicios comparables a

los de una máquina de mucho mayor tamaño. El coste de estas máquinas individuales más la red de alta velocidad necesaria para conectarlas puede ser menor que el de una supercomputadora de mayor precio, teniendo además una mayor fiabilidad y unos menores costes de mantenimiento. Dichos sistemas distribuidos se utilizan para proporcionar una **alta disponibilidad** (porque resulta mucho más probable que al menos uno de los miembros del *cluster* sea capaz de responder a una solicitud, incluso si hay otros miembros del *cluster* que han fallado o que no están disponibles) y un **equilibrado de carga**, porque la carga de trabajo puede desplazarse automáticamente desde los miembros del *cluster* que tengan demasiadas tareas que llevar a cabo a otros miembros que puedan estar mucho más descargados. El término **computación en retícula** hace referencia a sistemas distribuidos que están acoplados de forma más débil que los clusters, pero que siguen funcionando de manera conjunta para llevar a cabo tareas de gran envergadura. Los sistemas de computación en retícula pueden necesitar software especializado para que resulte más sencillo distribuir datos y algoritmos a las máquinas que participan en la retícula. Como ejemplos podemos citar el sistema Condor de la universidad de Wisconsin o el sistema BOINC (*Open Infrastructure for Network Computing*, Infraestructura abierta para la computación en red) de Berkeley. Ambos sistemas se instalan a menudo en computadoras que se utilizan para otros propósitos, como por ejemplo los PC que utilizamos en el trabajo o en nuestro domicilio, pudiendo esas máquinas ofrecer voluntariamente potencia de computación a la cuadrícula cuando no se las esté empleando para alguna otra cosa. Gracias a la creciente conectividad de Internet, este tipo de computación distribuida en retícula de carácter voluntario ha permitido que millones de PC domésticos colaboren en problemas matemáticos y científicos enormemente complejos. La **computación en nube**, en la que una serie de enormes conjuntos de computadoras compartidas en la red pueden asignarse para ser utilizados por los clientes según sea necesario es la tendencia más reciente en el campo de los sistemas distribuidos. De forma bastante similar a cómo la generalización de las redes eléctricas metropolitanas a principios del siglo xx eliminó la necesidad de que las fábricas y empresas individuales mantuvieran sus propios generadores, Internet está haciendo posible que todo tipo de entidades confíen sus datos y sus cálculos a "la Nube", que en este caso hace referencia a los enormes recursos de computación ya disponibles en la red. Servicios tales como Elastic Compute Cloud de Amazon permiten a los clientes alquilar computadoras virtuales por horas, sin preocuparse de dónde está ubicado realmete el hardware de la computadora. Google Docs y Google Apps permiten a los usuarios colaborar en la generación de información o construir servicios web sin la necesidad de saber cuántas computadoras están trabajando en el problema o dónde se almacenan los datos relevantes. Los servicios de computación en nube proporcionan garantías razonables de fiabilidad y escalabilidad, aunque también suscitan preocupaciones acerca de la privacidad y la seguridad en un mundo en el que puede que ya no sepamos quién posee y opera las computadoras que utilizamos.

Cuestiones y ejercicios

1. ¿Qué es una red abierta?

2. Indique la diferencia entre puente y conmutador.
3. ¿Que es un encaminador?
4. Identifique algunas relaciones sociales que se ajusten al modelo cliente/servidor.
5. Identifique algunos de los protocolos utilizados en la sociedad.
6. Resuma la distinción entre computación en cluster y computación en retícula.

4.2 Internet

El ejemplo más notable de interred es **Internet**, que tiene su origen en una serie de proyectos de investigación que datan de principios de la década de 1960. El objetivo era desarrollar la capacidad de enlazar diversas redes de computadoras, para que pudieran funcionar como un sistema conectado que no se viera afectado en su operación global por desastres de escala local. Buena parte de este trabajo estaba patrocinado por el gobierno de Estados Unidos a través de la agencia DARPA (*Defense Advanced Research Projects Agency*, Agencia de Proyectos Avanzados de Investigación para la Defensa). A lo largo de los años, el desarrollo de Internet fue pasando de ser un proyecto patrocinado por el gobierno a un proyecto académico de investigación y actualmente es, en buena medida, un esfuerzo de carácter comercial que enlaza una combinación mundial de redes LAN, MAN y WAN que abarcan a millones de computadoras.

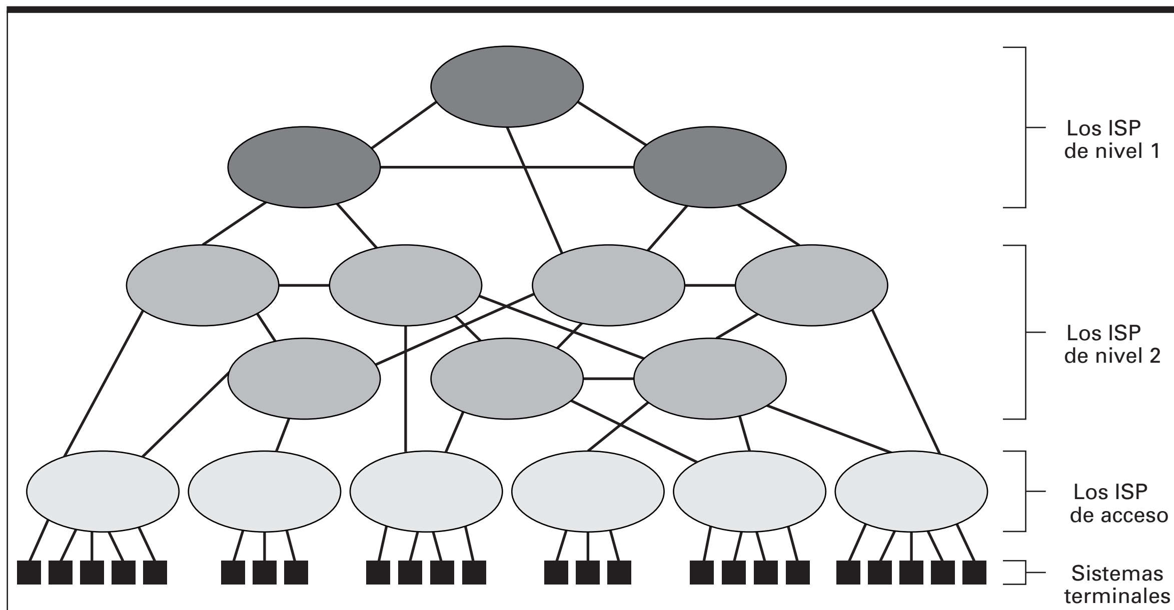
Arquitectura de Internet

Como ya hemos mencionado, Internet es un conjunto de redes conectadas. En general, estas redes son construidas y mantenidas por organizaciones denominadas **Proveedores de servicios de Internet** (ISP, *Internet Service Provider*). También resulta bastante común utilizar el término ISP para hacer referencia a las propias redes. Así, podemos utilizar expresiones como conectarse a un ISP, cuando lo que realmente queremos decir es conectarse a la red proporcionada por un ISP.

El sistemas de redes operado por los ISP puede clasificarse en una jerarquía de acuerdo con el papel que desempeñan en la estructura global de Internet (Figura 4.7). En la parte superior de esta jerarquía se encuentra un número relativamente pequeño de proveedores **ISP de nivel 1**, que están compuestos por redes WAN internacionales de alta velocidad y alta capacidad. Estas redes se consideran la red troncal de Internet. Normalmente, son operadas por grandes empresas que trabajan en el sector de las comunicaciones. Un ejemplo sería una empresa cuyo origen fuera el de una empresa de telefonía tradicional que hubiera expandido su ámbito de actuación para proporcionar otros servicios de comunicaciones.

Conectados a los ISP de nivel 1 se encuentran los **ISP de nivel 2**, que tienden a tener un ámbito más regional y que son menos potentes en lo que a sus capacidades se refiere (la distinción entre los ISP de nivel 1 y nivel 2 es, a menudo, una cuestión debatible). De nuevo, estas redes tienden a ser operadas por empresas que pertenecen al sector de las comunicaciones.

Figura 4.7 Composición de Internet.



Los ISP de nivel 1 y nivel 2 son básicamente redes de encaminadores que proporcionan de manera colectiva la infraestructura de comunicaciones de Internet. Teniendo en cuenta el papel que desempeñan podemos considerar que representan el auténtico núcleo de Internet. El acceso a este núcleo suele ser proporcionado por un intermediario denominado ISP de acceso. Un **ISP de acceso** es esencialmente una interred independiente, en ocasiones denominada **intranet**, operada por una única autoridad cuyo negocio consiste en suministrar acceso a Internet a los usuarios individuales. Como ejemplos podríamos citar empresas tales como AOL, Microsoft y las empresas locales de telefonía y cable que facturan por su servicio, además a algunas otras organizaciones como universidades o corporaciones, que se encargan de proporcionar acceso a Internet a los individuos que forman parte de sus organizaciones.

Los dispositivos con los que los usuarios individuales se conectan a los ISP de acceso se conocen con el nombre de **sistemas terminales**, **hosts** o **anfitriones**. Estos sistemas terminales no necesariamente son computadoras en el sentido tradicional de la palabra. Existe un amplio rango de dispositivos que pueden actuar como sistemas terminales incluyendo teléfonos, videocámaras, automóviles y electrodomésticos. Después de todo, Internet es esencialmente un sistema de comunicaciones, por lo que cualquier dispositivo que pudiera beneficiarse de la comunicación con otros dispositivos sería un sistema terminal en potencia.

La tecnología mediante la cual los sistemas terminales se conectan a los ISP de acceso también es muy variable. Quizá el tipo que más rápidamente está creciendo son las conexiones inalámbricas basadas en la tecnología WiFi. La estrategia consiste en conectar el punto de acceso de la red WiFi a un ISP de acceso, proporcionando así acceso a Internet a través de ese ISP a los sistemas terminales que se encuentran dentro del alcance de las emisiones del punto de acceso de

la red WiFi. El área de la zona de cobertura del punto de acceso de la red WiFi en la cual se proporciona acceso a Internet se suele denominar **punto caliente**. Los puntos calientes y las agrupaciones de puntos calientes son cada vez más comunes, pudiendo encontrarse en domicilios particulares, en hoteles, edificios de oficinas, pequeñas empresas, aparcamientos y en algunos casos, ciudades enteras. Una tecnología similar es la que se emplea en el sector de la telefonía móvil en la que los puntos calientes se conocen con el nombre de celdas y los "encaminadores" que generan las celdas se coordinan para proporcionar un servicio continuo, a medida que un sistema terminal se desplaza de una celda a otra.

Otras técnicas populares para la conexión con los ISP de acceso utilizan líneas telefónicas o sistemas por cable/satélite. Estas tecnologías pueden emplearse para proporcionar conexión directa a un sistema terminal o al encaminador de un cliente, al cual se conectarán múltiples sistemas terminales. Esta última táctica está siendo cada vez más popular en el mercado residencial, en el que se crea un punto caliente local en un domicilio mediante un encaminador/punto de acceso conectado a un ISP de acceso a través de las líneas telefónicas o de cable ya existentes.

Los enlaces por cable y vía satélite ya existentes son inherentemente más compatibles con la transferencia de datos a alta velocidad que las líneas telefónicas tradicionales, que se instalaron originalmente teniendo presentes las necesidades de la comunicación por voz. Sin embargo, se han desarrollado varios esquemas muy inteligentes para ampliar esos enlaces de voz con el fin de admitir la transmisión de datos digitales. Estas soluciones hacen uso de unos dispositivos denominados **modems** (modulador/demodulador) que convierten los datos digitales que hay transferir a un formato compatible con el medio de transmisión que se está utilizando. Un ejemplo sería **DSL** (***Digital Subscriber Line***, Línea digital de abonado) en donde se reserva el rango de frecuencias situado por debajo de 4 KHz (4 kilociclos por segundo) para la comunicación tradicional por voz, mientras que las frecuencias más altas se emplean para la transferencia de datos digitales. Otro enfoque más antiguo consiste en convertir los datos digitales en sonido y transmitir dicho sonido de la misma forma que se hace con la voz. Esta solución se conoce como acceso **telefónico**, en referencia al hecho de que se utiliza para conexiones temporales en las que el usuario establece una llamada telefónica tradicional con el encaminador del ISP y luego conecta su teléfono al sistema terminal que va a emplear. Aunque se trata de una solución barata y de alta disponibilidad, el acceso telefónico proporciona una tasa de transferencia de datos relativamente lenta, lo que hace que cada vez sea menos capaz de gestionar las aplicaciones Internet actuales, que tienden a depender de la comunicación de vídeo en tiempo real y de la transmisión de grandes bloques de datos. Debido a ello, un número creciente de pequeñas empresas y de domicilios particulares se conectan a sus ISP de acceso mediante tecnologías de banda ancha, incluyendo conexiones de televisión por cable, líneas telefónicas de datos dedicadas, antenas de comunicación vía satélite e incluso cables de fibra óptica.

Direccionamiento Internet

Como hemos visto en la Sección 4.1, una interred necesita un sistema de direccionamiento global que asigne una dirección unívoca de identificación a cada

Internet2

Ahora que Internet ha pasado de ser un proyecto de investigación a un producto de consumo doméstico, la comunidad científica ha pasado a centrar su atención en un proyecto denominado Internet2. Internet2 está pensado como un sistema únicamente académico e implica a numerosas universidades, que trabajan en colaboración con empresas y con el gobierno. El objetivo es llevar a cabo una serie de investigaciones en aplicaciones interred que requieren comunicación de banda ancha, como por ejemplo el control y acceso remoto a costosos equipos avanzados, como pueden ser telescopios y dispositivos de diagnóstico médico. Un ejemplo de esos proyectos actuales de investigación implica la cirugía remota realizada por una serie de manos robóticas que remedan las manos de un cirujano distante que observa al paciente por vídeo. Si desea obtener más información acerca de Internet2 consulte http://www.internet2.org.

computadora del sistema. En Internet, estas direcciones se conocen con el nombre de **direcciones IP**; el término *IP* hace referencia a "Internet Protocol" (Protocolo Internet), que es un término sobre el que hablaremos en la Sección 4.4. Originalmente, cada dirección IP era un patrón de 32 bits, pero para proporcionar un conjunto mayor de direcciones, actualmente está en marcha el proceso de conversión a direcciones de 128 bits (consulte las explicaciones acerca de IPv6 en la Sección 4.4). **ICANN** (*Internet Corporation for Assigned Names and Numbers*, Corporación Internet para la asignación de nombres y números), que es una organización sin ánimo de lucro establecida para coordinar el funcionamiento de Internet, se encarga de asignar a los ISP bloques de direcciones IP con numeración consecutiva. A los ISP se les permite entonces asignar las direcciones de los bloques que tienen concedidos a una serie de máquinas situadas dentro de su región de autoridad. De ese modo, se asignan direcciones IP unívocas a todas las máquinas de Internet .

Las direcciones IP se escriben tradicionalmente en **notación decimal con puntos** en la que los bytes de la dirección se separan mediante puntos y cada byte se expresa como un entero representado en notación tradicional de base diez. Por ejemplo, con la notación decimal con puntos, el patrón 5.2 representaría el patrón de dos bytes de longitud 0000010100000010, que está compuesto por el byte 00000101 (representado por el 5) seguido del byte 00000010 (representado por el 2). Por su parte, el patrón 17.12.25 representa un patrón de tres bytes de longitud compuesto por el byte 00010001 (que es 17 en notación binaria), seguido del byte 00001100 (12 en binario) y seguido del byte 00011001 (25 en binario). En resumen, una dirección IP de 32 bits podría aparecer escrita como 192.207.177.133 al ser expresada en notación decimal con puntos.

Las direcciones expresadas como un patrón de bits (incluso cuando se las comprime utilizando la notación decimal con puntos) no se prestan mucho a su manipulación por parte de los seres humanos. Por esta razón, Internet dispone de un sistema de direccionamiento alternativo en el que las máquinas se identifican mediante nombres mnemónicos. Este sistema de direccionamiento está basado en el concepto de **dominio**, que puede considerarse como una "región" de Internet operada por una única autoridad, como por ejemplo una universidad, un club, una empresa o un organismo gubernamental (ponemos entre

comillas la palabra región porque, como pronto veremos, ese tipo de región puede no corresponderse con un área física de Internet). Es necesario registrar cada dominio ante el ICANN, que es un proceso que se encargan de realizar una serie de empresas denominadas **registradoras**, a las que el ICANN ha asignado dicho papel. Como parte de este proceso de registro, a cada dominio se le asigna un **nombre de dominio** mnemónico, que es único entre todos los nombres de dominio de Internet. Los nombres de dominio suelen ser descriptivos de la organización que está registrando dicho dominio, lo que maximiza la utilidad de estos nombres para los seres humanos.

Por ejemplo, el nombre de dominio de la editorial Addison-Wesley es `aw.com`. Observe el sufijo situado a continuación del punto. Ese sufijo se emplea para reflejar la clasificación del dominio, que en este caso es "comercial", como indica el sufijo `com`. Estos sufijos se denominan **dominios de nivel superior** (TLD, *Top-Level Domain*). Otros TLD serían `edu` para instituciones educativas, `gov` para organismos gubernamentales de Estados Unidos, `org` para organizaciones sin ánimo de lucro, `museum` para museos, `info` para uso no restringido y `net`, que originalmente estaba pensado para los ISP pero que ahora se utiliza de forma mucho más general. Además de estos TLD generales, también están los TLD de dos letras para países específicos (denominados **TLD de código de país**), como por ejemplo `au` para Australia y `ca` para Canadá.

Una vez registrado el nombre mnemónico de un dominio, la organización que ha registrado dicho nombre es libre de ampliarlo para obtener identificadores mnemónicos para determinados elementos individuales dentro del dominio. Por ejemplo, una máquina individual dentro de Addison-Wesley podría estar identificada como `ssenterprise.aw.com`. Observe que los nombres de dominio se amplían hacia la izquierda y que las ampliaciones se separan mediante un punto. En algunos casos, se emplean múltiples extensiones, denominadas **subdominios**, como medio de organizar los nombres dentro de un dominio. Estos subdominios representan a menudo diferentes redes dentro de la jurisdicción del dominio. Por ejemplo, si a la Universidad Desconocida se le asignara el nombre de dominio `desconocida.edu`, entonces una computadora individual en la Universidad Desconocida podría tener un nombre como `r2d2.medicina.desconocida.edu`, lo que significa que la computadora `r2d2` se encuentra en el subdominio `medicina` dentro del dominio `desconocida` del TLD `edu`. (Hay que hacer hincapié que la notación con puntos utilizada en las notaciones mnemónicas no guarda ninguna relación con la notación decimal con puntos empleada para representar las direcciones en formato de patrón de bits.)

Aunque las direcciones mnemónicas resultan cómodas para los seres humanos, los mensajes siempre se transmiten a través de Internet mediante direcciones IP. Así, si un ser humano quiere enviar un mensaje a una máquina distante e identifica el destino por medio de una dirección mnemónica, el software que esté utilizando tiene que ser capaz de convertir dicha dirección en una dirección IP antes de transmitir el mensaje. Esta conversión se realiza con la ayuda de numerosos servidores, denominados **servidores de nombres**, que son esencialmente directorios que proporcionan servicios de traducción de direcciones a los clientes. Colectivamente, estos servidores de nombres se utilizan como un sistema de directorios global conocido como **Sistema de nombres de dominio** (DNS, *Domain Name System*). El proceso de utilizar el sistema DNS para realizar una traducción se denomina búsqueda **DNS**.

De esta forma, para que una máquina pueda ser accesible por medio de un nombre de dominio mnemónico, dicho nombre debe estar representado en alguno de los servidores de nombres del DNS. En aquellos casos en los que la entidad que ha establecido el dominio dispone de los recursos necesarios, puede establecer y mantener también su propio servidor de nombres que contenga todos los nombres pertenecientes a dicho dominio. De hecho, este es el modelo en el que se basaba originalmente el sistema de nombres de dominio. Cada dominio registrado representaba una región física de Internet, que era operada por algún tipo de autoridad local, como por ejemplo una empresa, una universidad o un organismo gubernamental. Esta autoridad era, en esencia, un ISP de acceso que proporcionaba a sus miembros acceso a Internet por medio de su propia intranet, la cual estaba conectada a Internet. Como parte de este sistema, la organización mantenía su propio servidor de nombres, que facilitaba los servicios de traducción para todos los nombres utilizados dentro de su dominio.

Este modelo sigue siendo común hoy día. Sin embargo, muchas empresas o pequeñas organizaciones quieren tener presencia en Internet y disponer de su propio dominio, pero sin dedicar los recursos necesarios para darle soporte. Por ejemplo, podría ser beneficioso para un club de ajedrez local tener presencia en Internet mediante el nombre de dominio `KingsandQueens.org`, pero es bastante probable que el club no disponga de los recursos necesarios para establecer su propia red, para mantener un enlace desde dicha red a Internet y para implementar su propio servidor de nombres. En este caso, el club puede firmar un contrato con un ISP de acceso para crear la apariencia de un dominio registrado utilizando los recursos que ya ha puesto en marcha el ISP. Normalmente, el club, quizá con la ayuda del ISP, registrará el nombre que el club haya seleccionado y firmará un contrato con el ISP para que dicho nombre se incluya en el servidor de nombres del ISP. Esto significa que todas las búsquedas DNS relativas al nuevo nombre de dominio serán dirigidas al servidor de nombres del ISP del que se obtendrá la traducción adecuada. De esta forma, en un mismo ISP pueden residir muchos dominios registrados, dándose a menudo el caso de que cada uno de ellos ocupa solamente una pequeña porción de una misma computadora.

Aplicaciones de Internet

En este apartado veremos algunas aplicaciones de Internet, comenzando con tres aplicaciones ***tradicionales***. Sin embargo, estas aplicaciones "convencionales" no consiguen mostrar adecuadamente todo el interés que Internet suscita en la actualidad. De hecho, la distinción entre una computadora y otros dispositivos electrónicos va siendo cada vez más difusa. Los teléfonos, las televisiones, los aparatos de reproducción de sonido, las alarmas antirrobo, los hornos microondas y las videocámaras son, todos ellos, "dispositivos Internet" en potencia. A su vez, las aplicaciones tradicionales de Internet están viéndose empequeñecidas por una marea en expansión de nuevas aplicaciones, incluyendo la mensajería instantánea, la videoconferencia, la telefonía Internet y la radio Internet. Después de todo, Internet es simplemente un sistema de comunicaciones por el que pueden transferirse datos. A medida que la tecnología continúa incrementando las tasas de transferencia de dicho sistema, el contenido de los datos que se transmiten solo está limitado por nuestra propia ima-

ginación. Por tanto, incluiremos también en nuestro análisis dos aplicaciones Internet más recientes: la telefonía y las difusiones por radio, para ilustrar algunas de las cuestiones asociadas con la Internet emergente hoy en día, incluyendo la necesidad de estándares de protocolo adicionales, la necesidad de enlazar Internet con otros sistemas de comunicaciones y la necesidad de ampliar la funcionalidad de los encaminadores de Internet.

Correo electrónico Uno de los usos más populares de Internet es el correo electrónico (***email***, abreviatura de ***electronic mail***), que es un sistema mediante el que se transfieren mensajes entre usuarios de Internet. Para proporcionar servicio de correo electrónico, la autoridad local de un dominio puede designar a una máquina concreta dentro de su dominio con el fin de que desempeñe el papel de **servidor de correo**. Normalmente, los servidores de correo se establecen dentro de los dominios operados por los ISP de acceso, con el fin de proporcionar servicio de correo a los usuarios pertenecientes a su ámbito de actuación. Cuando un usuario envía un correo electrónico desde su máquina local, el correo se transmite primero al servidor de correo del usuario. Desde allí, se reenvía al servidor de correo de destino, donde se almacena hasta que el receptor contacte con el servidor de correo y pida ver los mensajes de correo acumulados.

El protocolo utilizado para transferir correo entre servidores de correo, así como para enviar un nuevo mensaje desde la máquina local de su autor hasta el servidor de correo correspondientes es **SMTP** (***Simple Mail Transfer Protocol***, Protocolo simple de transferencia de correo). Puesto que SMTP se diseñó inicialmente para transferir mensajes de texto codificados con ASCII, se han desarrollado protocolos adicionales como **MIME** (***Multipurpose Internet Mail Extensions***, Extensiones multipropósito de correo Internet), con el fin de convertir los datos no ASCII a un formato compatible con SMTP.

Existen dos protocolos populares que pueden utilizarse para acceder al correo electrónico que se haya acumulado en el servidor de correo de un usuario. Estos protocolos son **POP3** (***Post Office Protocol version 3***, Protocolo de oficina de correo versión 3) e **IMAP** (***Internet Mail Access Protocol***, Protocolo de acceso a correo Internet). POP3 es el más simple de los dos. Utilizando POP3, el usuario transfiere los mensajes (los descarga) a su computadora local, en la que puede leerlos, almacenarlos en diversas carpetas, editarlos o manipularlos como desee. Todas estas tareas se efectúan en la máquina local del usuario usando el almacenamiento masivo de dicha máquina local. IMAP permite al usuario almacenar y manipular los mensajes y los materiales relacionados en la misma máquina que reside el servidor de correo. De esta forma, un usuario que tenga que acceder a su correo desde diferentes computadoras puede mantener sus mensajes en el servidor de correo, con lo que podrá acceder a ellos desde cualquier computadora remota a la que tenga acceso.

Teniendo en mente el papel de un servidor de correo, es fácil comprender la estructura de la dirección de correo electrónico de una persona. Está formada por una cadena de símbolos (que en ocasiones se denomina nombre de cuenta) que identifica a esa persona seguida del símbolo @, seguido de la cadena mnemónica que identifica al servidor de correo que debe recibir el mensaje. En realidad, a menudo identifica tan solo el dominio de destino, y el servidor de correo del dominio se identifica, en último término, por medio de

una búsqueda DNS. Así, la dirección de correo electrónico de una persona que trabaje en Addison-Wesley Inc. podría ser `shakespeare@aw.com`. En otras palabras, un mensaje enviado a esta dirección iría al servidor de correo del dominio `aw.com` donde sería almacenado para que los consulte la persona identificada por la cadena de símbolos `shakespeare`.

El protocolo de transferencia de archivos Una forma de transferir archivos (como documentos, fotografías o cualquier otra información codificada) es adjuntándolos a mensajes de correo electrónico. Sin embargo, un método más eficiente consiste en aprovechar el protocolo **FTP** (*File Transfer Protocol*, Protocolo de transferencia de archivos), que es un protocolo cliente/servidor que permite transferir archivos a través de Internet. Para trasferir un archivo utilizando FTP, un usuario en una computadora de Internet utiliza un paquete software que implementa FTP con el fin de establecer contacto con otra computadora. (La computadora original desempeña el papel de cliente y la computadora con la que contacta hace el papel de servidor, al que a menudo se denomina servidor FTP.) Una vez establecida esta conexión, pueden transferirse archivos entre ambas computadoras en ambas direcciones.

FTP se ha convertido en una forma muy popular de proporcionar acceso limitado a datos a través de Internet. Por ejemplo, suponga que queremos permitir que ciertas personas extraigan un archivo, al mismo tiempo que prohibimos acceso a todas las demás personas. Lo único que hace falta es colocar el archivo en una máquina con un servidor FTP y proteger el acceso al archivo mediante una contraseña. Así, las personas que conozcan la contraseña podrán obtener acceso al archivo mediante FTP, mientras que las restantes personas serán bloqueadas. A las máquinas de Internet utilizadas de este modo se las denomina en ocasiones sitios FTP, porque constituyen una ubicación dentro de Internet en la que hay archivos disponibles mediante FTP.

Los sitios FTP también se utilizan para proporcionar acceso no restringido a archivos. Para llevar esto a cabo, los servidores FTP utilizan el término ***anonymous*** como nombre de inicio de sesión universal. Dichos sitios se denominan con frecuencia sitios **FTP anónimos** y proporcionan acceso no restringido a los archivos almacenados en ellos.

Aunque los clientes y los servidores FTP están ampliamente disponibles, la mayoría de los usuarios satisface ahora sus necesidades de transferencia de archivos mediante exploradores web que utilizan el protocolo HTTP (del que hablaremos en la siguiente sección).

Telnet y SSH Uno de los primeros usos de Internet era permitir a los usuarios de computadoras acceder a esas computadoras a larga distancia. **Telnet** es un sistema de protocolo que fue definido precisamente con este objetivo. Utilizando telnet, un usuario (que ejecute un software de cliente telnet) puede contactar con el servidor telnet situado en una computadora distante y luego seguir el procedimiento de inicio de sesión de ese sistema operativo para obtener acceso a la máquina distante. Así, por medio de telnet, un usuario distante tiene el mismo acceso a las aplicaciones y utilidades residentes en una computadora que el que tendría un usuario local.

Habiendo sido diseñado durante las primeras etapas de desarrollo de Internet, telnet presenta varias carencias. Una de las más críticas, es que la comunicación por medio de telnet no está cifrada. Esto tiene su importancia

incluso si el contenido de la comunicación no es confidencial, ya que la contraseña del usuario forma parte de la comunicación que se establece durante el proceso de inicio de sesión. Como consecuencia, el uso de telnet abre la puerta para que alguien que esté espiando las comunicaciones pueda interceptar una contraseña y luego dar un mal uso a esa información crítica. Una alternativa a telnet que ofrece una solución a este problema y que está sustituyendo a telnet rápidamente es **SSH** (***Secure Shell***, Shell seguro). Entre las características de SSH podemos resaltar que se encarga de cifrar los datos que se están transfiriendo, así como de realizar la autenticación (Sección 4.5), que es el proceso de asegurarse de que las dos partes que se están comunicando son, de hecho, quienes dicen ser.

VoIP Como ejemplo de una de las más recientes aplicaciones de Internet, consideremos **VoIP** (***Voice over Internet Protocol***, Voz sobre IP), en la que se utiliza la infraestructura de Internet para proporcionar comunicaciones de voz similares a las de los sistemas de telefonía tradicionales. En su forma más simple, VoIP consiste en dos procesos situados en diferentes máquinas que transfieren datos de audio mediante el modelo P2P, un proceso que no presenta en sí mismo ningún problema significativo. Sin embargo, otras tareas, como la iniciación y recepción de llamadas, el enlace de VoIP con los sistemas de telefonía tradicionales y la provisión de servicios como por ejemplo llamadas a emergencias, son cuestiones que van más allá de las aplicaciones de Internet tradicionales. Además, los gobiernos que son propietarios de las empresas telefónicas tradicionales de su país ven VoIP como una amenaza y han prohibido su uso o han establecido fuertes impuestos sobre el mismo.

Los sistemas VoIP existentes se clasifican en cuatro tipos distintos, que están compitiendo entre sí para ver quién gana la carrera de la popularidad. Los **teléfonos software** VoIP están compuestos por un software P2P que permite que dos o más PC compartan una llamada, sin utilizar ningún hardware especializado, más allá de un altavoz y un micrófono. Un ejemplo de sistema de telefonía software VoIP es Skype, que también proporciona a sus clientes enlaces con el sistema de comunicaciones telefónicas tradicional. Una desventaja de Skype es que se trata de un sistema propietario, por lo que buena parte de su estructura operativa no se conoce de manera pública. Esto significa que los usuarios de Skype deben confiar en la integridad del software de Skype sin que exista verificación por parte de terceros. Por ejemplo, para recibir llamadas, el usuario de Skype debe dejar su PC conectado a Internet y a disposición del sistema Skype, lo que significa que una parte de los recursos del PC puede ser utilizada para dar soporte a otras comunicaciones de Skype, sin que el propietario del PC sea consciente de ello, funcionalidad que ha generado una cierta resistencia por parte de los clientes.

Un segundo tipo de VoIP está compuesto por **adaptadores telefónicos analógicos**, que son dispositivos que permiten a un usuario conectar su teléfono tradicional al servicio telefónico proporcionado por un ISP de acceso. Esta opción suele comercializarse conjuntamente con los servicios Internet tradicionales y/o los servicios de televisión digital.

El tercer tipo de VoIP se presenta de la forma de teléfonos VoIP integrados, que son dispositivos que sustituyen a un teléfono tradicional por otro terminal telefónico conectado directamente a una red TCP/IP. Los teléfonos VoIP integra-

Generaciones de teléfonos móviles

En la última década, la tecnología de teléfonos móviles ha progresado desde los dispositivos portátiles simples de un único propósito a computadoras de mano complejas y multifunción. La red de telefonía móvil de primera generación transmitía señales analógicas de voz a través del aire de forma muy similar a la de los teléfonos tradicionales pero sin el cable de cobre que atraviesa la pared. En retrospectiva, llamamos a estos primeros sistemas telefónicos redes "1G" o de primera generación. La segunda generación utilizaba señales digitales para codificar la voz, proporcionando un uso más efectivo del espectro y permitiendo la transmisión de otras clases de datos digitales tales como los mensajes de texto. Las redes de telefonía de tercera generación (**"3G"**) proporcionan tasas de transferencia de datos más altas, permitiendo realizar llamadas móviles de vídeo y otras actividades que requieren un gran ancho de banda. Los objetivos de la red 4G incluyen tasas de transferencia de datos todavía más altas y una red totalmente basada en conmutación de paquetes y que utilizará el protocolo IP, lo que proporcionará a la nueva generación de teléfonos inteligentes capacidades que hoy día solo están disponibles para los PC que tienen habilitadas conexiones de banda ancha.

dos están siendo cada vez más comunes en las grandes empresas, muchas de las cuales están sustituyendo sus sistemas telefónicos tradicionales internos de hilo de cobre por sistemas VoIP a través de Ethernet con el fin de reducir los costes y mejorar la funcionalidad.

Por último, la siguiente generación de teléfonos inteligentes está diseñada para utilizar tecnología VoIP. Es decir, las generaciones anteriores de teléfonos móviles solo se comunicaban con la red de la compañía telefónica empleando los protocolos de dicha compañía. El acceso a Internet se obtenía mediante pasarelas entre la red de la operadora e Internet, en las cuales se convertían las señales al sistema TCP/IP. Sin embargo, la nueva red telefónica **4G** está diseñada para ser en su totalidad una red basada en IP, lo que significa que un teléfono 4G será, en esencia, simplemente otra computadora host conectada a la Internet global.

Radio Internet Otra aplicación Internet reciente es la transmisión de emisoras de radio, un proceso que se denomina ***webcasting*** por oposición al término ***broadcasting*** (multidifusión), porque las señales se transfieren a través de Internet en lugar de "por el aire". Para ser más precisos, la radio Internet es un ejemplo específico de **flujos de audio** (*streaming audio*), que es un término que hace referencia a la transferencia de datos de sonido en tiempo real.

En un análisis superficial, puede parecer que la radio Internet no requiere demasiada consideración especial. Podríamos pensar que una emisora se podría limitar a establecer un servidor que enviara los programas a cada uno de los clientes que lo solicitaran. Esta técnica se conoce con el nombre de **N-unidifusión** (*N-unicast*). (Para ser más precisos, el término ***unidifusión*** hace referencia al hecho de que un emisor envíe mensajes a un receptor, mientras que ***N-unidifusión*** hace referencia a la situación en la que un único emisor realiza varias unidifusiones simultáneas.) La técnica de N-unidifusión ha sido aplicada en la práctica, pero tiene la desventaja de que exige una gran cantidad de trabajo por parte del servidor de la emisora, además de por parte de los vecinos inmediatos de dicho servidor en Internet. De hecho, la N-unidifusión fuerza al servidor a

enviar mensajes individuales a cada uno de sus clientes en tiempo real, y todos estos mensajes tienen que ser reenviados por los vecinos de ese servidor.

La mayoría de las alternativas a la N-unidifusión representan intentos de aliviar este problema. Una de esas alternativas aplica el modelo P2P de una forma que recuerda en cierto modo a los sistemas de compartición de archivos. Es decir, una vez que un terminal ha recibido los datos, comienza a distribuir dichos datos a aquellos terminales que aún están esperando, lo que significa que buena parte del problema de distribución se transfiere desde el origen de los datos a los terminales.

Otra alternativa, denominada **multidifusión** (*multicast*), transfiere el problema de distribución a los encaminadores de Internet. Utilizando la multidifusión, un servidor transmite un mensaje a múltiples clientes por medio de una única dirección y confía en que los encaminadores de Internet reconozcan el significado de dicha dirección y generen y reenvíen copias del mensaje a los destinos apropiados. La única dirección usada en la multidifusión se denomina dirección de grupo y está identificada mediante un patrón inicial de bits específico. Los bits restantes se emplean para identificar la estación emisora, que en la tecnología de multidifusión se denomina grupo. Cuando un cliente quiere recibir los mensajes de una emisora concreta (quiere suscribirse a un grupo concreto), notifica su deseo al encaminador más cercano. Básicamente, lo que hace ese encaminador es reenviar dicho deseo a través de Internet, para que otros encaminadores sepan que tienen que comenzar a reenviar en la dirección de dicho cliente todos los mensajes futuros que contenga la dirección de grupo indicada. En resumen, cuando se utiliza la multidifusión, el servidor transmite una única copia de la emisión, independientemente del número de clientes que estén a la escucha, y es responsabilidad de los encaminadores realizar copias de esos mensajes según sea necesario y encaminarlos hacia los destinos apropiados. Observe, por tanto, que las aplicaciones que utilizan la multidifusión requieren que se amplíe la funcionalidad de los encaminadores de Internet más allá de lo que eran sus tareas asignadas originales. Este proceso de ampliación está siendo acometido actualmente.

Vemos entonces que la radio Internet, al igual que VoIP, está creciendo en popularidad al mismo tiempo que trata de establecer unas bases comunes de funcionamiento. No estamos seguros de qué es lo que nos reserva el futuro al respecto. Sin embargo, a medida que se continúan expandiendo las capacidades de la infraestructura de Internet, podemos estar seguros de que con ellas se desarrollarán nuevas aplicaciones de la técnica de ***webcasting***.

Hoy día ya hay disponibles dispositivos integrados y computadoras domésticas capaces de enviar vídeo de alta definición bajo petición a través de Internet. Una amplia gama de televisiones, reproductores de DVD/Blu-ray y consolas de juegos pueden ahora conectarse directamente a la red TCP/IP para seleccionar contenido multimedia de entre una amplia variedad de servidores tanto gratuitos como de abono.

Cuestiones y ejercicios

1. ¿Cuál es el propósito de los ISP de nivel 1 y de nivel 2? ¿Cuál es el propósito de los ISP de acceso?

2. ¿Qué es DNS?
3. ¿Qué patrón de bits está representado por 3.6.9 en notación decimal con puntos? Exprese el patrón de bits 0001010100011100 utilizando notación decimal con puntos.
4. ¿En qué forma es la estructura de la dirección mnemónica de una computadora de Internet (como `r2d2.medicina.desconocida.edu`) similar a una dirección postal tradicional? ¿Tienen las direcciones IP esta misma estructura?
5. Cite tres tipos de servidores que pueden encontrarse en Internet e indique qué hace cada uno de ellos.
6. ¿Por qué se considera que SSH es superior a telnet?
7. ¿Cuáles son las diferencias entre las técnicas de emisión radio vía Internet basadas en P2P y en multidifusión y la técnica basada en N-unidifusión?
8. ¿Qué criterios hay que tener en cuenta a la hora de seleccionar uno de los cuatro tipos de VoIP?

4.3 La World Wide Web

En esta sección vamos a centrarnos en una aplicación Internet que permite diseminar información multimedia a través de Internet. Está basada en el concepto de **hipertexto**, un término que originalmente hacía referencia a documentos de texto que contenían enlaces, denominados **hipervínculos**, a otros documentos. Hoy día, el hipertexto se ha ampliado para incluir también imágenes, audio y vídeo, y a causa de esta expansión de su ámbito de actuación en ocasiones se denomina **hipermedia**.

Cuando se emplea una interfaz gráfica de usuario, el lector de un documento de hipertexto puede seguir los hipervínculos asociados con el mismo apuntando y haciendo clic con su ratón. Por ejemplo, suponga que en un documento de hipertexto aparece la frase "La interpretación del Bolero de Maurice Ravel por la orquesta fue excepcional" y que el nombre *Maurice Ravel* está vinculado a otro documento, que quizá proporcione información acerca del compositor. Un lector podría elegir ver ese material asociado apuntando al nombre *Maurice Ravel* con el ratón y haciendo clic en el botón del ratón. Además, si se instalan los hipervínculos apropiados, el lector podría escuchar una grabación de audio del concierto haciendo clic en la palabra *Bolero.*

De esta forma, un lector de documentos de hipertexto puede explorar documentos relacionados o seguir la línea de pensamiento de un documento a otro. A medida que se vinculan partes de documentos con otros documentos se va desarrollando una especie de tela de araña de información relacionada. Cuando se implementa en una red de computadoras este tipo de sistema, los documentos que componen esa "tela de araña" pueden residir en diferentes máquinas, formando una red de documentos global. Esa red de documentos que ha evolucionado en Internet tiene alcance mundial y se conoce como **World Wide Web** (también se la denomina **WWW**, **W3** o simplemente **Web**). A un documento de hipertexto en la World Wide Web a menudo se le llama

El Consorcio World Wide Web

El Consorcio World Wide Web (W3C) fue formado en 1994 para promocionar la World Wide Web desarrollando estándares de protocolos (conocidos como estándares W3C). W3C tiene su sede en el CERN, el laboratorio de partículas de alta energía de Ginebra, Suiza. El CERN es donde se desarrolló el lenguaje de composición HTML original, así como el protocolo HTTP para transferir documentos HTML a través de Internet. Hoy día, W3C es la fuente de muchos estándares (incluyendo estándares para XML y numerosas aplicaciones multimedia) que permiten la compatibilidad entre una amplia gama de productos Internet. Puede obtener más información acerca de W3C en su sitio web en la dirección http://www.w3c.org.

página web. Un conjunto de páginas web estrechamente relacionadas se conoce con el nombre de **sitio web**.

La World Wide Web tiene su origen en el trabajo de Tim Berners-Lee, que se dio cuenta del potencial que presentaba la combinación del concepto de documentos vinculados con la tecnología de interredes y que desarrolló el primer software para implementar la WWW en diciembre de 1990.

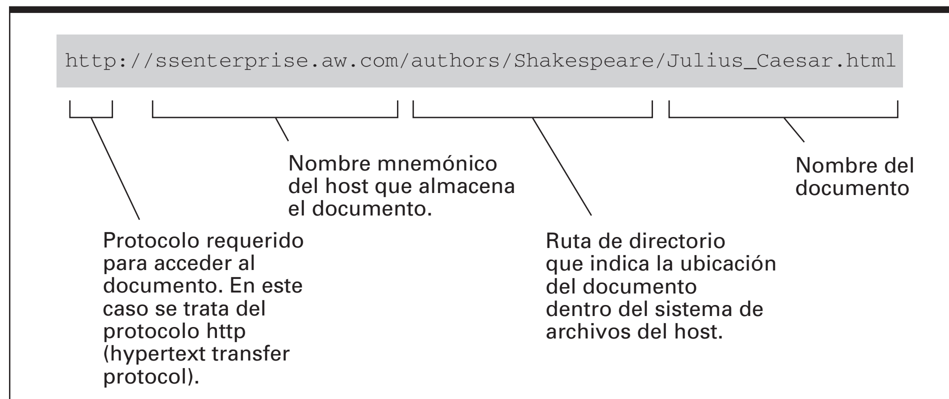
Implementación de la Web

Los paquetes software que permiten a los usuarios acceder a hipertexto en Internet caen en una de dos categorías: paquetes que desempeñan el papel de clientes y paquetes que desempeñan el papel de servidores. Un paquete de cliente reside en la computadora del usuario y se encarga de la tarea de obtener los materiales solicitados por el usuario y presentárselos de una forma organizada. Es el cliente que proporciona la interfaz de usuario el que permite a los usuarios navegar a través de la Web. De ahí que a ese cliente se le denomine a menudo navegador o **explorador** (*browser*), o en ocasiones explorador web. El paquete servidor (a menudo denominado **servidor web**) reside en una computadora que contiene los documentos de hipertexto a los que se puede acceder. Su tarea consiste en proporcionar acceso controlado a los documentos, según los vayan solicitando los clientes. En resumen, un usuario obtiene acceso a los documentos de hipertexto por medio de un explorador que reside en su computadora. Este explorador, que desempeña el papel de cliente, obtiene dichos documentos solicitando los servicios de servidores web dispersos por todo Internet. Normalmente, los documentos de hipertexto se transfieren entre los exploradores y los servidores web utilizando el protocolo **HTTP** (*Hypertext Transfer Protocol*, Protocolo de transferencia de hipertexto).

Para poder localizar y extraer documentos en la World Wide Web, a cada documento se le asigna una dirección unívoca denominada **URL** (*Uniform Resource Locator*, Localizador uniforme de recursos). Cada URL contiene la información necesaria para que un explorador contacte con el servidor apropiado y solicite el documento deseado. Así, para ver una página web, una persona proporciona a su explorador el URL del documento deseado y luego le ordena que lo extraiga y lo muestre.

En la Figura 4.8 se muestra un URL típico. Está formado por cuatro segmentos: el protocolo que hay que utilizar para comunicarse con el servidor que

Figura 4.8 Un URL típico.



controla el acceso al documento, la dirección mnemónica de la máquina que contiene al servidor, la ruta de directorios necesaria para que el servidor encuentre el directorio que contiene el documento y el nombre del propio documento. En resumen, el URL de la Figura 4.8 indica al explorador que debe contactar con el servidor web que se encuentra en la computadora conocida con el nombre de `ssenterprise.aw.com` utilizando el protocolo HTTP y extraer el documento de nombre `Julius_Caesar.html`, que se encuentra dentro del subdirectorio `Shakespeare`, que a su vez está dentro del directorio de nombre `authors`.

En ocasiones, una dirección URL puede no contener explícitamente todos los segmentos mostrados en la Figura 4.8. Por ejemplo, si el servidor no necesita seguir una ruta de directorio para acceder al documento, no se incluirá dicha ruta en el URL. Además, en ocasiones, un URL estará compuesto únicamente por un protocolo y la dirección mnemónica de una computadora. En estos casos, el servidor web situado en esa computadora devolverá un documento predeterminado, que normalmente se denomina página de inicio y que suele describir la información disponible en dicho sitio web. Dichas direcciones URL acortadas proporcionan un medio simple de contactar con las organizaciones. Por ejemplo, el URL `http://www.google.com` llevará a la página de inicio de Google, que contiene hipervínculos a los servicios, productos y documentos relacionados con dicha empresa.

Para simplificar aún más la localización de dichos sitios web, muchos exploradores suponen que será necesario emplear el protocolo HTTP si no se identifica ningún protocolo. Estos exploradores extraerán correctamente la página de inicio de Google cuando se les proporciona un "URL" compuesto meramente por `www.google.com`.

HTML

Un documento de hipertexto tradicional es similar a un archivo de texto, porque su texto se codifica carácter a carácter utilizando un sistema como ASCII o Unicode. La diferencia es que un documento de hipertexto también contiene símbolos especiales, denominados **etiquetas** (*tag*), que describen cómo debe aparecer el documento al visualizarlo en pantalla, qué recursos multimedia

(como por ejemplo imágenes) deben acompañar al documento y qué elementos del documento están vinculados a otros documentos. Este sistema de marcadores se conoce con el nombre de **HTML** (***Hypertext Markup Language***, Lenguaje de etiquetas de hipertexto).

Así, es en términos del lenguaje HTML como el autor de una página web describe la información que un explorador necesita para presentar esa página en la pantalla del usuario y para encontrar cualesquiera documentos relacionados a los que haga referencia en la página actual. El proceso es análogo a añadir instrucciones de maquetación en un texto mecanografiado simple (quizá utilizando un lápiz rojo), para que el maquetador sepa cómo debe aparecer el material en su forma final. En el caso del hipertexto, esas marcas rojas se sustituyen por etiquetas HTML y es el explorador quien desempeña en último término el papel del maquetador, leyendo las etiquetas HTML para saber cómo hay que presentar el texto en la pantalla de la computadora.

En la Figura 4.9a se muestra la versión codificada en HTML (el código **fuente**) de una página web extremadamente simple. Observe que las etiquetas están limitadas por los símbolos < y >. El documento fuente HTML está compuesto por dos secciones: una cabecera (delimitada por las etiquetas `<head>` y `</head>`) y un cuerpo (delimitado por las etiquetas `<body>` y `</body>`). La diferencia entre la cabecera y el cuerpo de una página web es similar a la que existe entre la cabecera y el cuerpo de un memorando de oficina típico. En ambos casos, la cabecera contiene la información preliminar acerca del documento (fecha, tema, etc. en el caso del memorando). El cuerpo contiene la sustancia del documento, que en el caso de una página web es el material que será presentado en la pantalla de la computadora cuando se muestre la página.

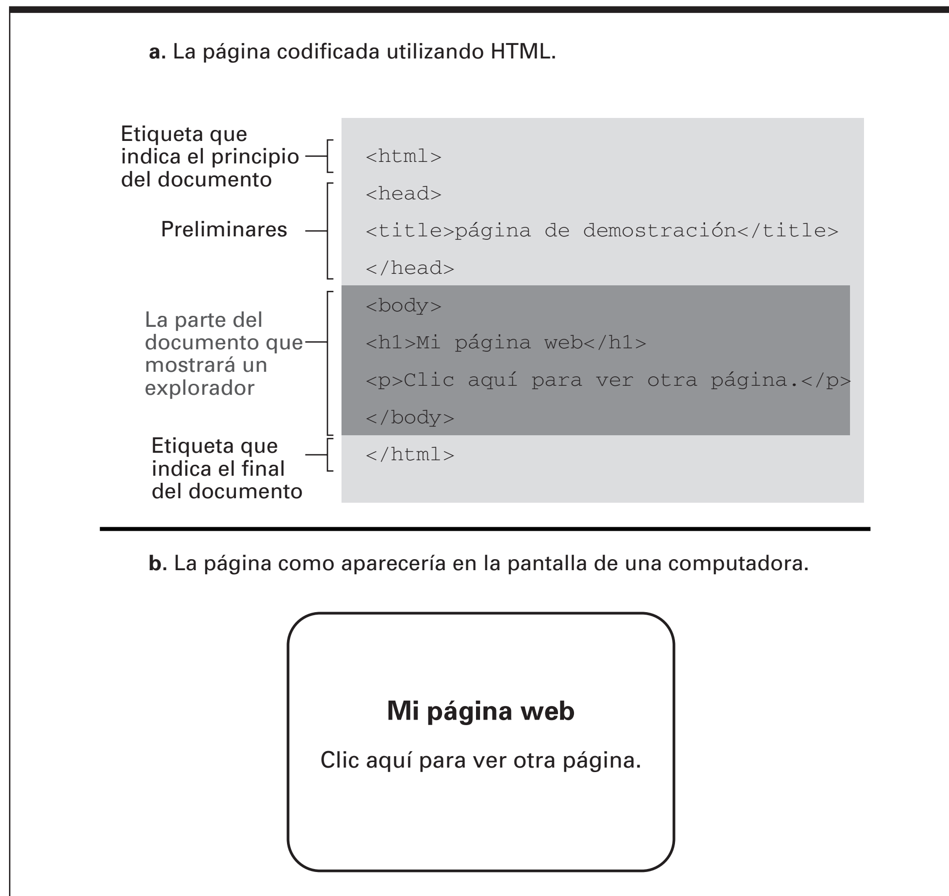
La cabecera de la página de la Figura 4.9a contiene únicamente el título del documento (delimitado por las etiquetas de título `<title>` y `</title>`). Este título es solo para propósitos de documentación; no forma parte de la página que hay que mostrar en la pantalla de la computadora. El material que se visualiza en la pantalla está contenido en el cuerpo del documento.

El primer elemento en el cuerpo del documento de la Figura 4.9a es un encabezado de nivel uno (delimitado por las etiquetas `<h1>` y `</h1>`) que contiene el texto "Mi página web". El hecho de que sea un encabezado de nivel uno significa que el explorador mostrará este texto de manera prominente en la pantalla. El siguiente elemento del cuerpo es un párrafo de texto (delimitado por las etiquetas `<p>` y `</p>`) que contiene el texto "Clic aquí para ver otra página.". La Figura 4.9b muestra la página tal como un explorador la mostraría en la pantalla de una computadora.

En su forma actual, la página de la Figura 4.9 no es completamente funcional en el sentido de que no sucederá nada cuando el usuario haga clic en la palabra ***aquí***, aunque la página lleve a pensar que el hacer clic provocaría que el explorador mostrara otra página. Para conseguir que se realice la acción apropiada, debemos vincular la palabra ***aquí*** con otro documento.

Supongamos que, cuando se hace clic sobre la palabra ***aquí***, queremos que el explorador extraiga y muestre la página situada en la dirección URL `http://crafty.com/demo.html`. Para ello, debemos en primer lugar rodear la palabra ***aquí*** en el código fuente de la página con las etiquetas `<a>` y `</a>`, que se denominan etiquetas de ancla. Dentro de la etiqueta de ancla de apertura insertaremos el parámetro

Figura 4.9 Una página web simple.




```
href = http://crafty.com/demo.html
```

(como se muestra en la Figura 4.10a), que indica que la referencia de hipertexto (`href`) asociada con la etiqueta es el URL situado a continuación del signo de igualdad (`http://crafty.com/demo.html`). Habiendo añadido las etiquetas de ancla, la página web aparecerá ahora en la pantalla de una computadora tal como se muestra en la Figura 4.10b. Observe que esta visualización es idéntica a la de la Figura 4.9b excepto porque la palabra ***aquí*** está resaltada indicando que se trata de un vínculo a otro documento web. Al hacer clic sobre esos términos resaltados, el explorador extrae y muestra el documento web asociado. Por tanto, es por medio de las etiquetas de ancla que pueden vincularse entre sí documentos web.

Por último, debemos indicar cómo podemos incluir una imagen en nuestra página web. Con este objetivo, supongamos que una codificación JPEG de la imagen que deseamos incluir está almacenada con el nombre de archivo `Image.jpg` en el directorio `Images` en `Images.com` y está disponible en el servidor web que se encuentra en esa ubicación. En estas condiciones, podemos decir al explorador que muestre la imagen en la parte superior de la pagina web

Figura 4.10 Una página web simple mejorada.

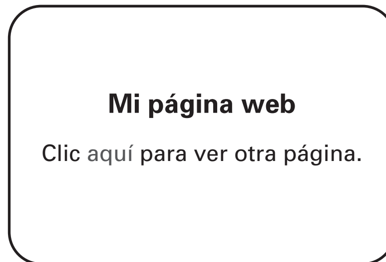
a. La página codificada utilizando HTML.

```
<html>
<head>
<title>Página de demostración</title>
</head>
<body>
<h1>My Web Page</h1>
<p>Clic
   <a href="http://crafty.com/demo.html">
   aquí
   </a>
   para ver otra página.</p>
</body>
</html>
```

Etiqueta de ancla que contiene un parámetro (señala la línea)

Etiqueta de ancla de cierre (señala la línea)

b. La página como aparecería en la pantalla de una computadora.



insertando la etiqueta de imagen <img src = "http://Images.com/Images/Image.jpg"> inmediatamente después de la etiqueta <body>, en el documento fuente HTML. Esto indica al explorador que debe mostrar al principio del documento la imagen de nombre Image.jpg. (La palabra src es una abreviatura de "source" (origen), lo que quiere decir que la información que hay después del signo de igualdad indica el origen de la imagen que hay que mostrar.) Cuando el explorador encuentra esta etiqueta, envía un mensaje al servidor HTTP de Images.com solicitando la imagen denominada Image.jpg y luego la muestra apropiadamente.

Si moviéramos la etiqueta de imagen al final del documento justo antes de la etiqueta </body>, entonces el explorador mostraría la imagen en la parte inferior de la pagina web. Por supuesto, existen técnicas más sofisticadas para posicionar una imagen en una página web, pero por el momento no necesitamos preocuparnos por ellas.

XML

HTML es básicamente un sistema de notación mediante el que puede codificarse un documento de texto junto con la apariencia del mismo, en forma de un simple archivo de texto. De forma similar, también podemos codificar material no textual en forma de archivos de texto, como por ejemplo una partitura musical. A primera vista, el patrón de pentagramas, compases y notas con el que tradicionalmente se representa la música no se adapta al formato carácter a carácter impuesto por los archivos de texto. Sin embargo, podemos solventar este problema desarrollando un sistema de notación alternativo. Para ser más precisos, podríamos acordar representar el principio de un pentagrama mediante <pentagrama clave = "sol">, el final del pentagrama mediante </pentagrama>, el tempo con la forma <tempo> 2/4 </tempo>, el principio y el final de cada compás mediante <compas> y </compas>, respectivamente, una nota como por ejemplo un Do corchea como <notas> Do corchea </notas>, y así sucesivamente. Entonces, el texto

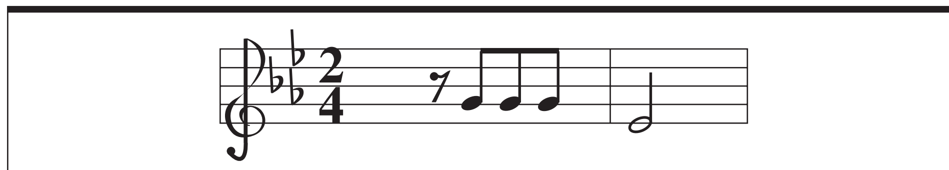
```
<pentagrama clave = "sol"> <clave>Do menor</clave>
<tempo> 2/4 </tempo>
<compas> <silencio> corchea </silencio> <notas> Sol
corchea, Sol corchea, Sol corchea </notas></compas>
<compas> <notas> Mi blanca </notas></compas>
</pentagrama>
```

podría utilizarse para codificar la partitura mostrada en la Figura 4.11. Utilizando esta notación, se podrían codificar, modificar, almacenar y transferir pentagramas a través de Internet en forma de archivos de texto. Además, podría escribirse software para presentar el contenido de tales archivos en el formato tradicional de las partituras o incluso para reproducir la música en un sintetizador.

Observe que nuestro sistema de codificación de partituras emplea el mismo estilo que el lenguaje HTML. Hemos elegido delimitar las etiquetas que identifican los distintos componentes mediante los símbolos < y >. Hemos decidido indicar el principio y el final de las estructuras (como por ejemplo un pentagrama, una secuencia de notas o un compás) mediante etiquetas del mismo nombre, estando el marcador de cierre designado mediante una barra inclinada (un <compas> se termina con la etiqueta </compas>). Y hemos decidido indicar los atributos especiales dentro de las etiquetas mediante expresiones tales como clave = "sol". Este mismo estilo podría emplearse también para desarrollar sistemas que permitan representar otros formatos como por ejemplo expresiones matemáticas y gráficos.

El lenguaje **XML** (***eXtensible Markup Language***, Lenguaje de composición extensible) es un estilo estandarizado (similar al de nuestro ejemplo musical)

Figura 4.11 Los dos primeros compases de la 5.ª Sinfonía de Beethoven.



para diseñar sistemas de notación para la representación de datos mediante archivos de texto. (En realidad, XML es un derivado simplificado de un conjunto más antiguo de estándares denominado Lenguaje generalizado estándar de composición, más conocido por sus siglas SGML, *Standard Generalized Markup Language*.) De acuerdo con el estándar XML, se han desarrollado sistemas de notación denominados **lenguajes de composición** para representar expresiones matemáticas, presentaciones multimedia y música. De hecho, HTML es el lenguaje de composición basado en el estándar XML que se desarrolló para representar páginas web. Realmente, la versión original de HTML fue desarrollada antes de que se solidificara el estándar XML, por lo que algunas características de HTML no se adaptan estrictamente a XML. Esta es la razón de que podamos encontrarnos con referencias a XHTML, que es la versión de HTML que se ajusta de forma rigurosa a XML.

XML proporciona un buen ejemplo de cómo se diseñan los estándares para tener un amplio rango de aplicaciones. En lugar de diseñar lenguajes de composición individuales, no relacionados entre sí, para codificar diversos tipos de documentos, el enfoque representado por XML consiste en desarrollar un estándar para los lenguajes de composición en general. Con este estándar, pueden desarrollarse lenguajes de composición para distintas aplicaciones. Los lenguajes de composición desarrollados de esta manera tienen una uniformidad que permite combinarlos para obtener lenguajes de composición para aplicaciones complejas como por ejemplo para documentos de texto que contengan segmentos de partituras junto con expresiones matemáticas.

Finalmente, es preciso observar que XML permite el desarrollo de nuevos lenguajes de composición que difieran de HTML en el sentido de que pongan el énfasis en la semántica, en lugar de en la apariencia. Por ejemplo, con HTML los ingredientes de una receta pueden componerse para que aparezcan en forma de una lista, en la que cada ingrediente se coloca en una línea separada. Pero si utilizáramos etiquetas con orientación semántica, los ingredientes de una receta podrían marcarse simplemente como ingredientes (quizá usando las etiquetas `<ingrediente>` y `</ingrediente>`) en lugar de como simples elementos en una lista. La diferencia es bastante sutil pero muy importante. El enfoque semántico permitiría que los **motores de búsqueda** (sitios web que ayudan a los usuarios a localizar material web relacionado con un tema de su interés) identificaran las recetas que contienen o no contienen ciertos ingredientes, lo que representaría una mejora sustancial con respecto al estado actual de las cosas, en el que solo es posible localizar aquellas recetas que contengan o no ciertas palabras. Para ser más precisos, si se emplearan etiquetas semánticas, un motor de búsqueda podría identificar recetas de lasaña que no contengan espinacas, mientras que una búsqueda similar basada simplemente en las palabras que forman el contenido se saltaría una receta que comenzara con la frase "Esta lasaña no contiene espinacas." A su vez, utilizando un estándar global de Internet para la composición de documentos de acuerdo con su semántica, en lugar de su apariencia, podría crearse una World Wide Web *semántica*, en lugar de la World Wide Web *sintáctica* que tenemos hoy día.

Actividades en el lado del cliente y en el lado del servidor

Considere ahora los pasos que harían falta para que un explorador accediera a la página web simple de la Figura 4.10 y la mostrara en la pantalla de la com-

putadora en la que el explorador se está ejecutando. En primer lugar, desempeñando el papel de cliente, el explorador utilizaría la información de una dirección URL (quizá obtenida de la persona que está usando el explorador) para contactar con el servidor web que controla el acceso a esa página y pedirle que le transfiera una copia de la misma. El servidor respondería enviando el documento de texto mostrado en la Figura 4.10a al explorador. Este interpretaría entonces las etiquetas HTML del documento para determinar cómo debe visualizarse la página y presentaría el documento correspondiente en la pantalla de su computadora. El usuario del explorador vería una imagen como la mostrada en la Figura 4.10b. Si el usuario hiciera entonces clic con el ratón sobre la palabra ***aquí***, el explorador utilizaría el URL de la etiqueta de ancla asociada para contactar con el servidor apropiado con el fin de obtener y mostrar otra página web. En resumen, el proceso consiste simplemente en que el explorador vaya extrayendo y mostrando sucesivas páginas web bajo el control del usuario.

Pero, ¿qué sucede si queremos ver una página web que contiene una animación o que permite a un cliente rellenar un formulario de pedido y enviar dicho pedido? Estas necesidades requerirían actividades adicionales por parte del explorador o del servidor web. Dichas actividades se denominan actividades del **lado del cliente** si son realizadas por un cliente (como por ejemplo un explorador) o actividades del **lado del servidor** si son realizadas por un servidor (como por ejemplo un servidor web).

Por ejemplo, suponga que una agencia de viajes quiere que los clientes puedan identificar los destinos y las fechas de viaje deseados, en cuyo momento la agencia presentará al cliente una página web personalizada con únicamente la información que sea pertinente para las necesidades de ese cliente. En este caso, el sitio web de la agencia de viajes proporcionaría primero una página web que mostraría al cliente los destinos disponibles. Basándose en esta información, el cliente especificaría los destinos de su interés y las fechas en las que desea viajar (una actividad del lado del cliente). Esta información sería entonces transferida al servidor de la agencia, donde se utilizaría para construir la página web personalizada apropiada (una actividad del lado del servidor), que después sería enviada al explorador del cliente.

Otro ejemplo es el que tiene lugar cuando utilizamos los servicios de un motor de búsqueda. En este caso, el usuario de la máquina cliente especifica un tema de su interés (una actividad del lado del cliente), que es transferida a continuación al motor de búsqueda, en el que se construye una página web personalizada que identifica los documentos que posiblemente le interesen al usuario (una actividad del lado del servidor), después de lo cual se envía al cliente esa página personalizada. Otro ejemplo más sería el caso del **correo web**, un medio cada vez más popular que los usuarios de computadoras utilizan para acceder a su correo electrónico a través de exploradores web. En este caso, el servidor web es un intermediario entre el cliente y el servidor de correo del mismo. Esencialmente, el servidor web construye páginas web que contienen información procedente del servidor de correo (una actividad del lado del servidor) y envía dichas páginas a la máquina del cliente donde el explorador las presenta (una actividad del lado del cliente). A la inversa, el explorador permite al usuario crear mensajes (una actividad del lado del cliente) y envía dicha información al servidor web, que luego reenvía los men-

sajes al servidor de correo (una actividad del lado del servidor) para que este los encamine hacia su destino final.

Existen numerosos sistemas para la realización de actividades del lado del cliente y del lado del servidor, compitiendo todos ellos por ver cuál consigue ser más prominente. Una técnica ya antigua pero muy popular de controlar las actividades del lado del cliente consiste en incluir programas escritos en el lenguaje JavaScript (desarrollado por Netscape Communications, Inc.) dentro del documento que contiene el código fuente HTML de la página web. Con ello, un explorador puede extraer esos programas y ejecutarlos según sea necesario. Otra técnica (desarrollada por Sun Microsystems) consiste en transferir primero una página web a un explorador y luego transferir unidades de programa adicionales denominadas ***applets*** (escritas en lenguaje Java) a ese explorador según se vaya solicitando desde dentro del documento fuente HTML. Otra tercera técnica es el sistema Flash (desarrollado por Macromedia) mediante el cual pueden implementarse presentaciones multimedia complejas del lado del cliente.

Un método antiguo de controlar las actividades del lado del servidor consistía en utilizar un conjunto de estándares denominados CGI (***Common Gateway Interface***, Interfaz común de pasarela) mediante el que los clientes podían solicitar la ejecución de programas almacenados en un servidor. Una variante de esta técnica (desarrollada por Sun Microsystems) consiste en permitir que los clientes provoquen la ejecución en el lado del servidor de unidades de programa denominadas ***servlets***. Puede aplicarse una versión simplificada de la técnica basada en servlets cuando la actividad del lado del servidor que se solicita es la construcción de una página web personalizada, como sucedía en nuestro ejemplo de la agencia de viajes. En este caso, se almacenan en el servidor web unas plantillas de página web denominadas JavaServer Pages (JSP) y esas plantillas se completan utilizando la información recibida desde el cliente. Microsoft utiliza una técnica similar, en la que las plantillas a partir de las cuales se construyen las páginas web personalizadas se denominan páginas ASP (Active Server Pages). A diferencia de estos sistemas propietarios, PHP (que originalmente significaba ***Personal Home Page***, página de inicio personal, pero que ahora se considera que significa PHP ***Hypertext Processor***, procesador de hipertexto PHP) es un sistema de código abierto para la implementación de funcionalidad en el lado del servidor.

Por último, pecaríamos de negligentes si no llamáramos la atención sobre los problemas de seguridad y los problemas éticos que surgen al permitir a los clientes y servidores ejecutar programas en sus máquinas interlocutoras. El hecho de que los servidores web transfieran programas de forma rutinaria a las máquinas cliente, donde se los ejecuta, hace que se planteen una serie de cuestiones éticas en el lado del servidor y una serie de problemas de seguridad en el lado del cliente. Si el cliente ejecuta ciegamente cualquier programa que le envíe un servidor web, estará abriendo la puerta de su máquina a la ejecución de actividades maliciosas por parte del servidor. De la misma forma, el hecho de que los clientes puedan hacer que se ejecuten programas en el lado del servidor plantea una serie de cuestiones éticas en el lado del cliente y una serie de cuestiones de seguridad en el lado del servidor Si el servidor ejecutara ciegamente cualquier programa que un cliente le envía, podrían aparecer fallos de seguridad y producirse daños en el propio servidor.

Cuestiones y ejercicios

1. ¿Qué es un URL? ¿Qué es un explorador?
2. ¿Qué es un lenguaje de composición?
3. ¿Cuál es la diferencia entre HTML y XML?
4. ¿Cuál es el propósito de cada una de las siguientes etiquetas HTML?
   a. `<html>`
   b. `<head>`
   c. `</p>`
   d. `</a>`
5. ¿A qué se refieren los términos ***lado del cliente*** y ***lado del servidor***?

4.4 Protocolos Internet

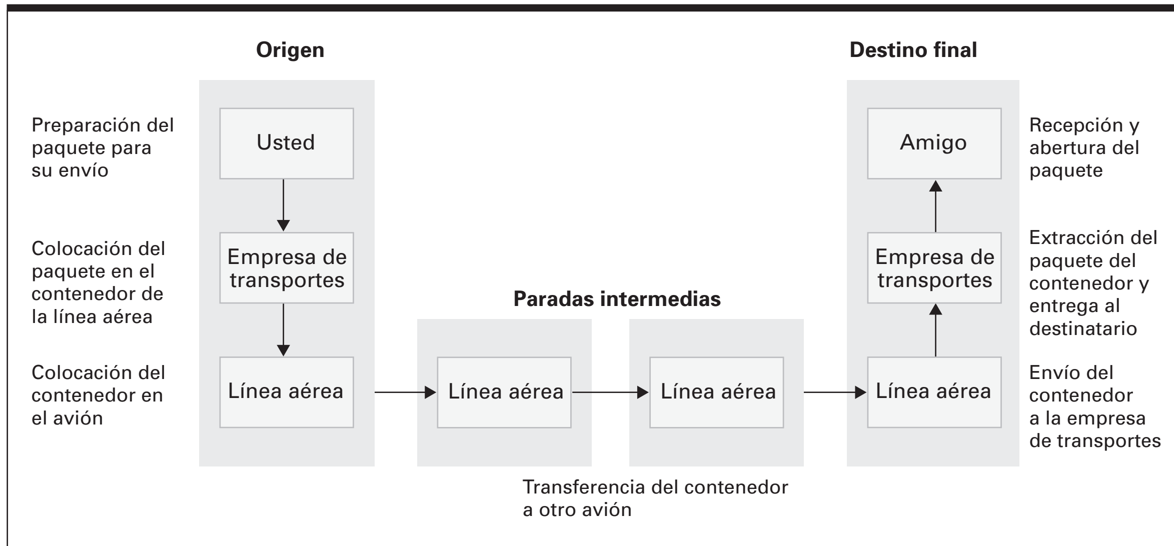
En esta sección vamos a investigar cómo se transfieren los mensajes a través de Internet. Este proceso de transferencia requiere la cooperación de todas las computadoras del sistema, por lo que en cada computadora de Internet hay disponible un software encargado de controlar este proceso. Vamos a comenzar estudiando la estructura global de este software.

División en capas del software de Internet

Una de las tareas principales del software de red consiste en proporcionar la infraestructura necesaria para transferir mensajes de una máquina a otra. En Internet, esta actividad de paso de mensajes se lleva a cabo por medio de una jerarquía de unidades software, que realiza tareas análogas a aquellas que se realizarían si quisiéramos enviar un paquete de regalo desde Estados Unidos a Europa (Figura 4.12). El primer paso consistiría en envolver el regalo en un paquete y escribir la dirección apropiada en el exterior del mismo. Después llevaríamos el paquete a una empresa de transportes o a un servicio postal. La empresa de transportes podría colocar el paquete junto con otros muchos en un gran contenedor y entregar dicho contenedor a una línea área cuyos servicios haya contratado. La línea aérea colocaría el contenedor en un avión y lo llevaría hasta la ciudad de destino, quizá haciendo una serie de paradas intermedias a lo largo del camino. En el destino final, la línea aérea extraería el contenedor del avión y se lo entregaría a la oficina de la empresa de transportes existente en la ciudad de destino. A su vez, la empresa de transportes sacaría el paquete del contenedor y se lo entregaría al destinatario.

En resumen, el transporte del regalo sería realizado mediante una jerarquía de tres capas: (1) la capa de usuario (formada por usted y su amigo), (2) la empresa de transportes y (3) la línea aérea. Cada capa utiliza la siguiente capa de nivel inferior como una herramienta abstracta. (A usted no le preocupan los detalles de operación de la empresa de transportes y a la empresa de transportes no le preocupan los detalles internos de funcionamiento de la línea aérea.) Cada capa de la jerarquía tiene representantes tanto en el origen como en el

Figura 4.12 Ejemplo de envío de un paquete.



destino, y los representantes del destino tienden a hacer la operación inversa de la que han hecho sus equivalentes en el origen.

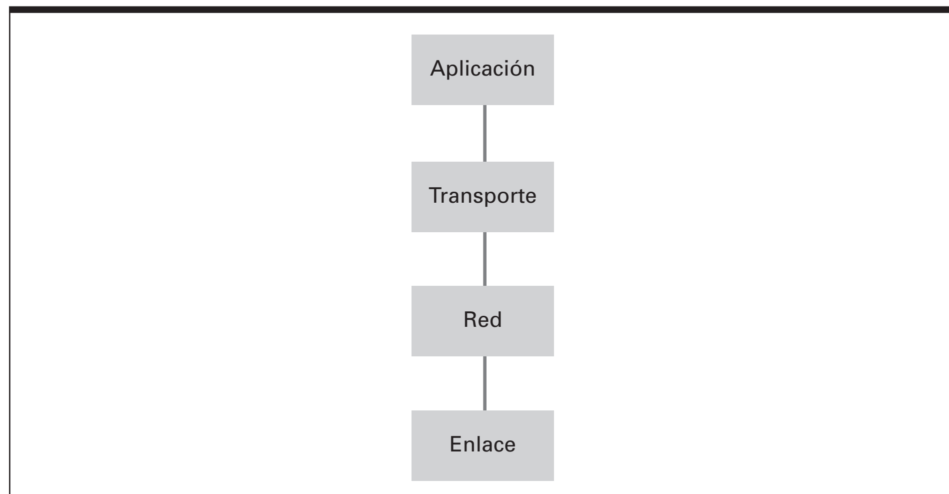
Lo mismo sucede con el software que controla las comunicaciones a través de Internet, excepto porque el software de Internet tiene cuatro capas en lugar de tres, cada una de ellas compuesta por un conjunto de rutinas software, en lugar de estar formadas por personas y empresas. Las cuatro capas se conocen con los nombres de **capa de aplicación**, **capa de transporte**, **capa de red** y **capa de enlace** (Figura 4.13). Los mensajes suelen originarse en la capa de aplicación. A partir de ahí se pasan hacia abajo a través de las capas de transporte y de red, mientras que se preparan para su transmisión y finalmente se envían por medio de la capa de enlace. El mensaje es recibido por la capa de enlace situada en el destino, después de lo cual se pasa a través de la jerarquía hacia arriba hasta que se entrega a la capa de aplicación situada en el destino del mensaje.

Vamos a investigar este proceso más en profundidad, siguiéndole la pista a un mensaje a medida que va circulando por el sistema (Figura 4.14). Comenzamos nuestro viaje en la capa de aplicación.

La capa de aplicación está compuesta por unidades software tales como los clientes y servidores que utilizan las comunicaciones Internet para llevar a cabo sus respectivas tareas. Aunque los nombres son similares, esta capa no está restringida a software perteneciente a la clase de las aplicaciones que hemos presentado en la Sección 3.2, sino que incluye también paquetes de utilidad. Por ejemplo, el software para transferir archivos mediante FTP o para proporcionar capacidades de inicio de sesión remota utilizando SSH se ha hecho tan común que ahora se considera normalmente software de utilidad.

La capa de aplicación emplea a la capa de transporte para enviar y recibir mensajes a través de Internet, de forma bastante similar a como nosotros empleamos a la empresa de transportes para enviar y recibir paquetes. Al igual que es responsabilidad nuestra proporcionar una dirección compatible con las

Figura 4.13 Las capas del software de Internet.



especificaciones de la empresa de transportes, es responsabilidad también de la capa de aplicación proporcionar una dirección que sea compatible con la infraestructura de Internet. Para satisfacer esta necesidad, la capa de aplicación puede utilizar los servicios de los servidores de nombres de Internet con el fin de traducir las direcciones mnemónicas empleadas por los seres humanos en direcciones IP compatibles con Internet.

Una tarea importante que lleva a cabo la capa de transporte es la de aceptar los mensajes procedentes de la capa de aplicación y asegurarse de que esos mensajes están adecuadamente formateados para su transmisión a través de Internet. Para conseguir este objetivo, la capa de transporte divide los mensajes largos en pequeños segmentos, que se transmiten a través de Internet en forma de unidades individuales. Esta división es necesaria porque un único mensaje de gran tamaño podría obstruir el flujo de otros mensajes en los encaminadores de Internet en los que se cruzan los caminos de numerosos mensajes. De hecho, los pequeños segmentos de mensajes pueden entrelazarse en esos puntos, mientras que un mensaje de gran tamaño forzaría a otros a esperar mientras que él pasa (de forma muy similar a como los automóviles tienen que esperar a que pase un tren de gran longitud en un paso a nivel).

La capa de transporte añade números de secuencia a los segmentos que genera para que esos segmentos puedan ser reordenados cuando lleguen al destino del mensaje. A continuación, entrega esos segmentos, que se conocen con el nombre de **paquetes**, a la capa de red. A partir de este punto, los paquetes se tratan como si fueran mensajes individuales no relacionados entre sí, hasta que llegan a la capa de transporte de su destino final. Es perfectamente posible que los diversos paquetes relacionados con un mismo mensaje sigan rutas diferentes a través de Internet.

Es tarea de la capa de red decidir a qué dirección hay que enviar cada paquete en cada uno de los pasos que componen la ruta que sigue el paquete a través de Internet. De hecho, la combinación de la capa de red y de la capa de enlace situada por debajo de ella es lo que constituye el software que reside en los encaminadores de Internet. La capa de red está a cargo de mantener la

Figura 4.14 Seguimiento de un mensaje a través de Internet.

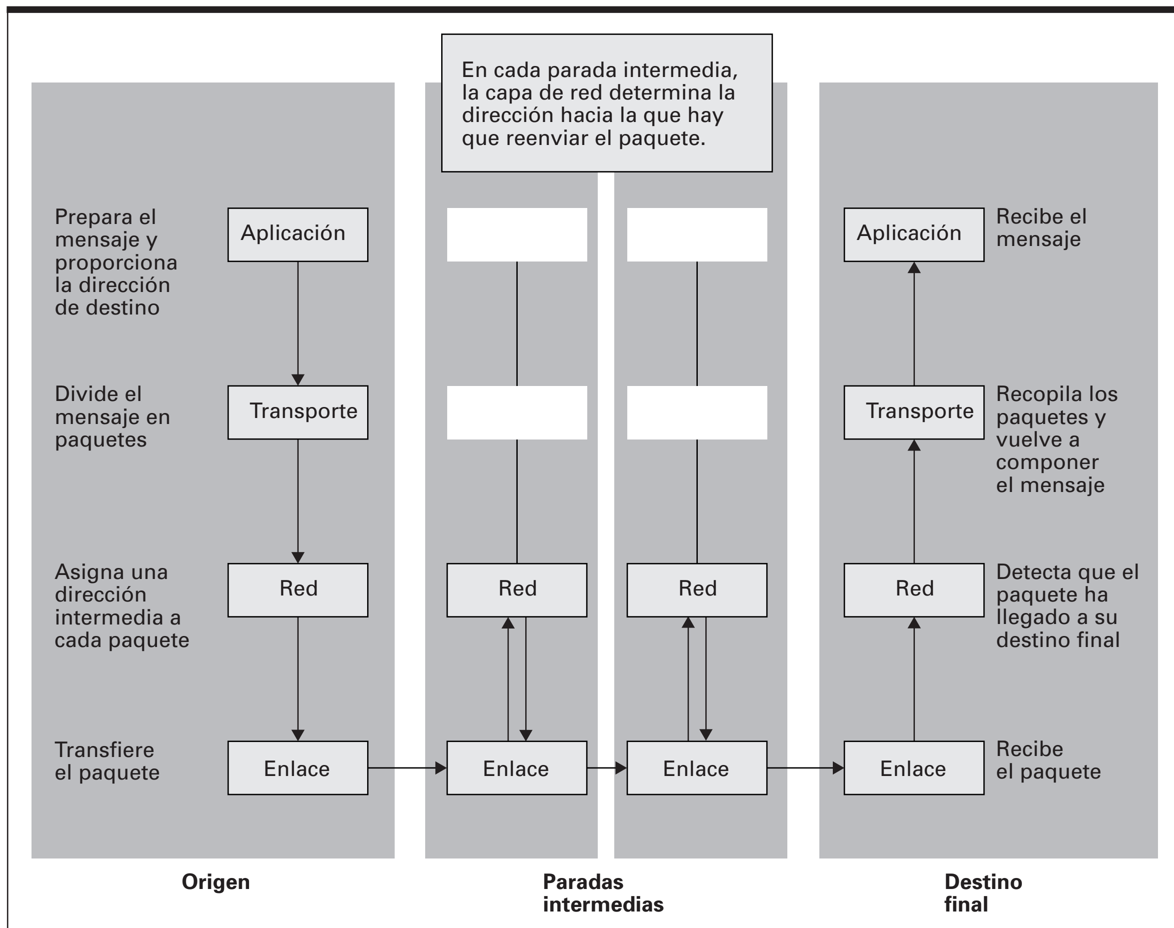


tabla de reenvío del encaminador y de utilizar dicha tabla para determinar la dirección en la que hay que reenviar los paquetes. La capa de enlace del encaminador es la que se encarga de recibir y transmitir los paquetes.

Así, cuando la capa de red situada en el origen de un paquete recibe el paquete desde la capa de transporte, utiliza su tabla de reenvío para determinar a dónde hay que enviar el paquete para que este inicie su viaje. Habiendo determinado la dirección adecuada, la capa de red entrega el paquete a la capa de enlace para su transmisión.

La capa de enlace tiene la responsabilidad de transferir el paquete. Por tanto, la capa de enlace debe encargarse de gestionar todos los detalles de las comunicaciones correspondientes a la red individual en la que reside la computadora. Por ejemplo, si se trata de una red Ethernet, la capa de enlace aplicará CSMA/CD. Si es una red WiFi, la capa de enlace aplicará CSMA/CA.

Cuando se transmite un paquete, este será recibido por la capa de enlace situada en el otro extremo de la conexión. Allí, la capa de enlace entrega el paquete a su capa de red, donde se compara el destino final del paquete con la tabla de reenvío contenida en la capa de red, con el fin de determinar cuál es la

dirección en la que se encuentra el siguiente paso del paquete. Tomada esta decisión, la capa de red devuelve el paquete a la capa de enlace, para que lo reenvíe en la dirección correcta. De esta manera cada paquete va saltando de máquina en máquina de camino a su destino final.

Observe que en las paradas intermedias de este viaje solo están involucradas las capas de enlace y de red (consulte de nuevo la Figura 4.14), por lo que estas son las únicas capas presentes en los encaminadores, como ya hemos mencionado. Además, para minimizar el retardo en cada una de estas "paradas", la tarea de reenvío que lleva a cabo la capa de red dentro de un encaminador está estrechamente integrada con la capa de enlace. Debido a ello, el tiempo que un encaminador moderno necesita para reenviar un paquete se mide en millonésimas de segundo.

En el destino final del paquete, es la capa de red la que reconoce que el viaje del paquete ha terminado. En ese caso, la capa de red entrega el paquete a su capa de transporte en lugar de reenviarlo de nuevo. A medida que la capa de transporte va recibiendo paquetes de la capa de red, extrae los segmentos de mensaje contenidos en estos paquetes y reconstruye el mensaje original de acuerdo con los números de secuencia incluidos por la capa de transporte situada en el origen del mensaje. Una vez recompuesto el mensaje, la capa de transporte se lo entrega a la unidad apropiada dentro de la capa de aplicación, completando así el proceso de transmisión del mensaje.

El determinar qué unidad dentro de la capa de aplicación debe recibir un mensaje entrante es una tarea de la capa de transporte. Esta tarea se lleva a cabo asignando **números de puerto** distintivos (y que no están relacionados con los puertos de E/S de los que hemos hablado en el Capítulo 2) a las diversas unidades y exigiendo que se añada el número de puerto apropiado a la dirección de cada mensaje antes de que este inicie su viaje. Entonces, una vez que el mensaje es recibido por la capa de trasporte del destino, esta simplemente entrega el mensaje a la unidad software de la capa de aplicación que tenga asignado el número de puerto correspondiente.

Los usuarios de Internet raramente necesitan preocuparse de los número de puerto, porque las aplicaciones más comunes tienen asignados unos números de puerto universalmente aceptados. Por ejemplo, si se le pide a un servidor web que extraiga un documento cuya dirección URL es `http://www.zoo.org/animales/frog.html`, el explorador asume que tiene que contactar con el servidor HTTP `www.zoo.org` situado en el número de puerto 80. De la misma forma, a la hora de transferir un archivo, un cliente FTP asume que tiene que comunicarse con el servidor FTP a través de los números de puerto 20 y 21.

En resumen, la comunicación a través de Internet implica la interacción de cuatro capas de software. La capa de aplicación gestiona los mensajes desde el punto de vista de una aplicación. La capa de transporte convierte esos mensajes en segmentos que sean compatibles con Internet y recompone los mensajes recibidos antes de entregarlos a la aplicación apropiada. La capa de red se ocupa de dirigir los segmentos a través de Internet. La capa de enlace se encarga de la propia transmisión de los segmentos de una máquina a otra. Teniendo en cuenta todas estas actividades, resulta hasta cierto punto sorprendente que el tiempo de respuesta de Internet se mida en milisegundos, de tal manera que muchas transacciones parecen tener lugar de manera instantánea.

El conjunto de protocolos TCP/IP

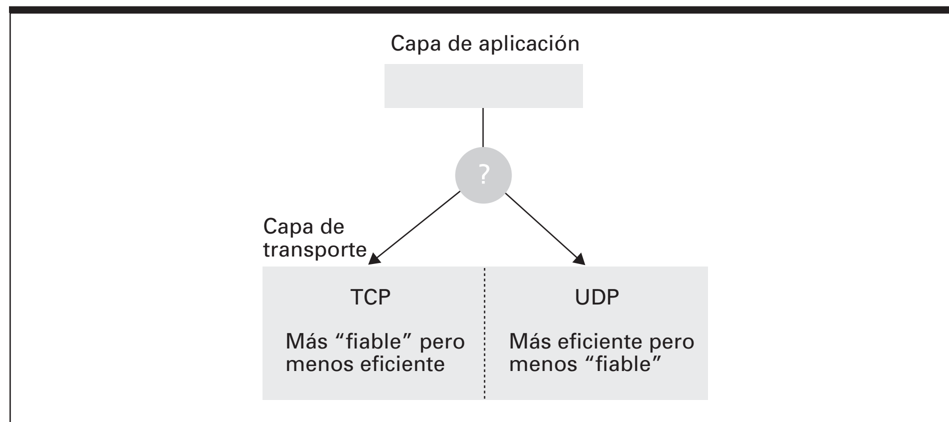
La demanda de redes abiertas ha generado la necesidad de que se publiquen estándares que permitan a los fabricantes suministrar equipos y software que funcionen apropiadamente con los productos de otros fabricantes. Uno de los estándares desarrollados con este objetivo es el modelo de referencia **OSI** (***Open System Interconnection***, Interconexión de sistemas abiertos), desarrollado por ISO (***International Organization for Standardization***). Este estándar está basado en una jerarquía de siete capas, en lugar de en la jerarquía de cuatro capas que acabamos de describir. Se trata de un modelo muy citado porque está avalado por la autoridad de una organización internacional, pero está tardando mucho en sustituir al modelo de cuatro capas, principalmente porque el modelo de referencia OSI fue establecido después de que la jerarquía de cuatro capas se hubiera convertido en el estándar de facto en Internet.

El conjunto de protocolos TCP/IP es un conjunto de estándares de protocolo utilizado en Internet para implementar la jerarquía de comunicaciones de cuatro capas. De hecho, el protocolo **TCP** (***Transmission Control Protocol***, Protocolo de control de transmisión) y el protocolo **IP** (***Internet Protocol***, Protocolo Internet) son los nombres de solo dos de los protocolos que componen este amplio conjunto, así que el hecho de que llamemos conjunto de protocolos TCP/IP al conjunto completo resulta un tanto confuso. Para ser más precisos, TCP define una versión de la capa de transporte. Decimos una ***versión*** porque el conjunto de protocolos TCP/IP proporciona más de una forma de implementar la capa de transporte; otra de las opciones está definida por **UDP** (***User Datagram Protocol***, Protocolo de datagramas de usuario). Esta diversidad es análoga al hecho de que, a la hora enviar un paquete, podemos elegir entre diferentes empresas de transportes, cada una de las cuales ofrece el mismo servicio básico pero con sus propias características distintivas. Por tanto, dependiendo de la calidad de servicio concreta que necesitemos, una unidad dentro de la capa de aplicación podría decidir enviar los datos a través de una versión TCP o UDP de la capa de transporte (Figura 4.15).

Existen varias diferencias entre TCP y UDP. Una de ellas es que, antes de enviar un mensaje a solicitud de la capa de aplicación, una capa de transporte basada en TCP envía su propio mensaje a la capa de transporte situada en el destino, informándole de que va a enviar un mensaje. A continuación, espera a que se confirme la recepción de este mensaje, antes de comenzar a enviar el mensaje de la capa de aplicación. De esta manera, decimos que la capa de transporte TCP establece una conexión antes de enviar un mensaje. Una capa de transporte basada en UDP no establece una conexión antes de enviar un mensaje, sino que se limita a enviar el mensaje en la dirección que se le ha indicado y se olvida de él. Por lo que a UDP respecta, podría suceder perfectamente que la computadora de destino ni siquiera estuviera operativa. Por esta razón, decimos que UDP es un protocolo no orientado a conexión.

Otra diferencia entre TCP y UDP es que las capas de transporte TCP situadas en el origen y el destino trabajan conjuntamente por medio de confirmaciones y retransmisiones de paquetes para asegurarse de que todos los segmentos de un mensaje se transfieran correctamente al destino. Por esta razón, decimos que TCP es un protocolo fiable, mientras que UDP, que no ofrece dichos servicios de retransmisión, es lo que denominamos un protocolo no fiable.

Figura 4.15 Elección entre TCP y UDP.



Otra diferencia más entre TCP y UDP es que TCP proporciona tanto un **control de flujo** (lo que quiere decir que la capa de transporte TCP situada en la máquina de origen de un mensaje puede reducir la velocidad con la que transmiten los segmentos, para evitar saturar a su corresponsal situado en el destino), como **control de congestión** (que quiere decir que la capa de transporte TCP situada en la máquina de origen de un mensaje puede ajustar su velocidad de transmisión para aliviar la congestión existente entre ella y la máquina de destino del mensaje).

Todo esto no quiere decir que UDP sea una mala elección. Después de todo, una capa de transporte basada en UDP es mucho más simple que otra basada en TCP, por lo que si una aplicación está dispuesta a aceptar las consecuencias potenciales del uso de UDP, puede que esta sea la mejor opción. Por ejemplo, la eficiencia de UDP hace que sea el protocolo preferido para las búsquedas DNS y para VoIP. Sin embargo, como el correo electrónico es mucho menos sensible al tiempo de transmisión, los servidores de correo emplean TCP para transferir sus mensajes.

IP es el estándar de Internet para implementar las tareas asignadas a la capa de red. Ya hemos indicado que estas tareas son el **reenvío**, que implica retransmitir los paquetes a través de Internet y el **encaminamiento**, que implica actualizar la tabla de reenvío contenida en la capa de red, con el fin de reflejar los cambios en las condiciones de la red. Por ejemplo, un encaminador puede estar funcionando mal, lo que significa que se debe dejar de reenviar tráfico en su dirección; o bien una sección de Internet puede congestionarse, lo que quiere decir que el tráfico debe encaminarse de manera que se evite esa zona bloqueada. Buena parte del estándar IP asociado con las tareas de encaminamiento se ocupa de los protocolos empleados para la comunicación entre capas de red vecinas, a medida que intercambian información de encaminamiento.

Una característica interesante asociada con el mecanismo de reenvío es que cada vez que una capa de red IP en la máquina de origen del mensaje prepara un paquete, le añade un valor denominado **contador de saltos** o tiempo de vida. Este valor es un límite que indica el número máximo de veces que el paquete debe ser reenviado mientras trata de encontrar su camino a través de Internet.

Cada vez que una capa de red IP reenvía un paquete, decrementa en una unidad el contador de saltos. Con esta información, la capa de red puede proteger a la propia red Internet evitando que haya paquetes circulando eternamente dentro del sistema. Aunque Internet continúa creciendo día a día, un valor inicial de 64 para el contador de saltos sigue siendo más que suficiente para permitir que un paquete encuentre su camino a través del laberinto de encaminadores de los ISP actuales.

Durante años se ha utilizado una versión de IP conocida con el nombre de IPv4 (IP versión cuatro) para implementar la capa de red en Internet. Sin embargo, Internet está sobrepasando rápidamente el sistema de direccionamiento de interred de 32 bits definido en IPv4. Para resolver este problema, así como para implementar otras mejoras tales como la multidifusión, se ha desarrollado una nueva versión de IP conocida con el nombre de IPv6, que emplea direcciones de interred de 128 bits. El proceso de conversión de IPv4 a IPv6 está actualmente en marcha (esta es la conversión a la que aludíamos en nuestra introducción a las direcciones de Internet en la Sección 4.2) y se espera que el uso de direcciones de 32 bits en Internet se haya extinguido en el año 2025.

Cuestiones y ejercicios

1. ¿Qué capas de la jerarquía del software de Internet no son necesarias en un encaminador?
2. ¿Cuáles son las diferencias entre una capa de transporte basada en el protocolo TCP y otra basada en el protocolo UDP?
3. ¿Cómo determina la capa de transporte qué unidad de la capa de aplicación debe recibir un mensaje entrante?
4. ¿Qué impide que una computadora de Internet almacene copias de todos los mensajes que pasan a su través?

4.5 Seguridad

Cuando una computadora está conectada a una red pasa a ser objetivo de accesos no autorizados y del vandalismo. En esta sección vamos a ocuparnos de los temas asociados con estos problemas.

Formas de ataque

Existen numerosas formas en las que una computadora y su contenido pueden ser atacados a través de las conexiones de red. Muchas de ellas incorporan el uso de software malicioso (conocido colectivamente como **malware**). Este software puede ser transferido a la propia computadora y ejecutado en ella, o bien puede atacar a una computadora a distancia. Entre los ejemplos de software que se transfiere y ejecuta en la propia computadora atacada se incluyen los virus, los gusanos, los caballos de Troya y el software espía (*spyware*), nombres que reflejan las principales características de cada uno de ellos.

El Equipo de Respuesta a Emergencias Informáticas

En noviembre de 1988, un gusano liberado en Internet causó una importante interrupción del servicio. En consecuencia, la agencia DARPA (Defense Advanced Research Projects Agency) de Estados Unidos creó el Equipo de Respuesta a Emergencias Informáticas (CERT, Computer Emergency Response Team), ubicado en el Centro de coordinación del CERT en la Universidad de Carnegie-Mellon. El CERT es la "policía" que se encarga de la seguridad en Internet. Entre sus responsabilidades están la investigación de problemas de seguridad, la emisión de alertas de seguridad y la implementación de campañas de concienciación pública para mejorar la seguridad de Internet. El Centro de coordinación del CERT mantiene un sitio web en la dirección http://www.cert.org donde publica noticias acerca de sus actividades.

Un **virus** es un software que infecta a una computadora insertándose a sí mismo en los programas que ya residen en la máquina. Después, cuando se ejecuta el programa "anfitrión", el virus también se ejecuta. Al ejecutarse, muchos virus lo único que hacen es prácticamente tratar de transferirse a sí mismos a otros programas dentro de la propia computadora. Sin embargo, algunos virus llevan a cabo acciones devastadoras, como corromper partes del sistema operativo, borrar grandes bloques de almacenamiento masivo o corromper de alguna otra forma los datos y otros programas.

Un **gusano** es un programa autónomo que se transfiere a sí mismo a través de una red, instalándose en distintas computadoras y reenviando copias de sí mismo a otras máquinas conectadas. Como en el caso de un virus, un gusano puede estar diseñado simplemente para replicarse a sí mismo o para llevar a cabo actividades vandálicas más serias. Una consecuencia característica de la operación de un gusano es que se produce una explosión en el número de copias replicadas del gusano, lo que hace que se degrade el rendimiento de las aplicaciones legítimas y puede terminar por sobrecargar toda una red completa o una interred.

Un **caballo de Troya** es un programa que entra en una computadora disfrazado como un programa legítimo, como por ejemplo un juego o un paquete de utilidad, siendo el propio usuario el que introduce a propósito el programa malicioso. Una vez en la computadora, el caballo de Troya realiza actividades adicionales que pueden tener efectos nocivos. En ocasiones, estas actividades se inician de forma inmediata. En otros casos, el caballo de Troya puede permanecer durmiente hasta verse activado por un suceso específico, como por ejemplo la llegada de una fecha preseleccionada. Los caballos de Troya llegan a menudo en forma de adjuntos a mensajes de correo electrónico incitantes. Cuando se abre el adjunto (es decir, cuando el receptor pide visualizar el adjunto), se activan las partes maliciosas del caballo de Troya. Por tanto, nunca deben abrirse los adjuntos de correo electrónico procedentes de fuentes desconocidas.

Otra forma de software malicioso es el **software espía** (*spyware* o también software *sniffing*), que es software que recopila información acerca de las actividades realizadas en la computadora en la que reside y que envía dicha información al instigador del ataque. Algunas empresas utilizan software espía como medio de recopilar perfiles de los clientes, y en este contexto ese tipo de

comportamiento es bastante cuestionable desde el punto de vista ético. En otros casos, el software espía se emplea con propósitos claramente maliciosos como registrar las secuencias de símbolos que se escriben en el teclado de la computadora, para buscar así contraseñas o números de tarjetas de crédito.

Por contraste con ese procedimiento de obtener información de manera secreta mediante software espía, el ***phishing*** es una técnica que consiste en obtener información explícitamente, limitándose a pedírsela al propio usuario. El término *phishing* es una derivación de la palabra inglesa *fishing* que significa pescar, porque esta técnica de recopilación de información confidencial consiste en echar numerosos "cebos", confiando en que alguien "pique". El *phishing* suele llevarse a cabo a través del correo electrónico y, cuando adopta esta forma, se parece bastante a las antiguas bromas telefónicas. El atacante envía mensajes de correo electrónico haciéndose pasar por una institución financiera, un organismo gubernamental o un grupo de las fuerzas de seguridad. En ese correo electrónico se pide a las víctimas potenciales una cierta información que se supone que se necesita para algún propósito legítimo. Sin embargo, la información así obtenida es utilizada por el atacante con propósitos hostiles.

A diferencia de las infecciones de carácter interno, como los virus y el software espía, una computadora de una red también puede ser atacada por software que se esté ejecutando en otras computadoras del sistema. Un ejemplo serían los ataques por **denegación de servicio** (DoS, *Denial of Service*), que es el proceso de sobrecargar una computadora con mensajes. En el pasado se han lanzado en Internet numerosos ataques por denegación de servicio contra servidores web comerciales de gran envergadura, con el fin de paralizar los negocios de la empresa; en algunos casos, esos ataques han provocado la completa detención de las actividades comerciales de esas empresas.

Un ataque por denegación de servicio requiere la generación de un gran número de mensajes en un corto periodo de tiempo. Para llevar esto a cabo, el atacante suele introducir software en un gran número de computadoras que no son conscientes de haber sido infectadas y que se dedicarán a generar mensajes cuando se les proporcione una señal. Entonces, cuando esa señal se produce, todas esas computadoras inundan al objetivo del ataque con una catarata de mensajes. Por tanto, en los ataques por denegación de servicio es imprescindible contar con un gran número de computadoras que son utilizadas como cómplices involuntarios en el ataque. Esta es la razón por la que se recomienda a todos los usuarios de PC que no dejen sus computadoras conectadas a Internet cuando no las estén utilizando. Según las estimaciones existentes, una vez que un PC se ha conectado a Internet, al menos un intruso tratará de aprovechar la existencia de ese PC en un lapso de 20 minutos. A su vez, un PC no protegido representa una significativa amenaza para la integridad de Internet.

Otro problema asociado con una abundancia de mensajes no deseados es la proliferación de correo electrónico basura (***spam***). Sin embargo, a diferencia de un ataque por denegación de servicio, el volumen de correo basura raramente es suficiente como para sobrecargar a una computadora. Más bien, el efecto de ese correo basura es que la persona que lo recibe termina hartándose. El problema se agrava por el hecho de que como, ya hemos visto, el correo basura es un medio muy utilizado para los ataques de *phishing* y para la propagación de caballos de Troya que pueden utilizarse para difundir virus y otros tipos de software malicioso.

Protección y remedios

Dice el viejo refrán que más vale prevenir que curar, y este refrán es completamente cierto en el contexto de la lucha por controlar las actividades vandálicas en las conexiones de red. Una de las principales técnicas de prevención consiste en filtrar el tráfico que pasa a través de un punto de la red, normalmente mediante un programa llamado **cortafuegos**. Por ejemplo, podemos instalar un cortafuegos en la pasarela de la intranet de una organización, con el fin de filtrar los mensajes que entran y salen de esa zona. Tales cortafuegos pueden estar diseñados para bloquear los mensajes salientes que tengan ciertas direcciones de destino o para bloquear los mensajes entrantes procedentes de orígenes que se sabe que pueden ser conflictivos. Esta última función es una herramienta que puede utilizarse con éxito para rechazar un ataque por denegación de servicio, porque proporciona un medio de bloquear el tráfico procedente de las computadoras que participan en el ataque. Otro papel común de los cortafuegos en una pasarela es el de bloquear todos los mensajes entrantes cuyas direcciones de origen se correspondan con la región a la que se accede a través de la pasarela, ya que tales mensajes indicarían que alguien está intentando, desde el exterior, hacerse pasar por un miembro de esa zona interna. La actividad consistente en hacerse pasar por alguien que uno no es se conoce con el nombre de **suplantación** (*spoofing*).

Los cortafuegos también se utilizan para proteger computadoras individuales en lugar de redes o dominios complejos. Por ejemplo, si no se está utilizando una computadora como servidor web, como servidor de nombres o como servidor de correo electrónico, debería instalarse un cortafuegos en dicha computadora para bloquear todo el tráfico entrante dirigido a ese tipo de aplicaciones. De hecho, una forma en la que un intruso puede conseguir acceder a una computadora es estableciendo contacto a través de un "agujero" dejado por un servidor inexistente. En particular, un método para extraer información recopilada por un software espía consiste en establecer un servidor clandestino en la computadora infectada, mediante el que una serie de clientes maliciosos pueden extraer lo que el software espía vaya averiguando. Un cortafuegos adecuadamente instalado puede bloquear los mensajes procedentes de estos clientes maliciosos.

Algunas variantes de cortafuegos están diseñadas para propósitos específicos, un ejemplo serían los **filtros de correo basura**, que son cortafuegos diseñados para bloquear el correo electrónico no deseado. Muchos filtros de correo basura utilizan técnicas bastante sofisticadas con el fin de distinguir entre correo electrónico deseable y correo electrónico basura. Algunos de esos programas aprenden a efectuar esta distinción mediante un proceso de entrenamiento durante el que el usuario identifica los ejemplos de correo basura hasta que el filtro adquiere suficientes datos como para tomar sus propias decisiones. Estos filtros son ejemplos de cómo diversas áreas de conocimiento (teoría de la probabilidad, inteligencia artificial, etc.) pueden contribuir conjuntamente a los desarrollos en otros campos.

Otra herramienta preventiva que se acerca al concepto de filtrado es el servidor proxy. Un **servidor proxy** es un software que actúa como intermediario entre un cliente y un servidor con el fin de proteger al cliente de las potenciales acciones adversas llevadas a cabo por el servidor. Sin un servidor proxy, un cliente se comunicaría directamente con el servidor, lo que quiere decir que el

servidor tiene la oportunidad de obtener cierta cantidad de datos acerca del cliente. Con el tiempo, a medida que múltiples clientes pertenecientes a la intranet de una organización se comunican con un servidor distante, dicho servidor puede recopilar una gran cantidad de información acerca de la estructura interna de la intranet (información que puede ser empleada posteriormente para llevar a cabo actividades maliciosas). Para protegerse frente a esta eventualidad, la organización puede establecer un servidor proxy para un tipo concreto de servicio (FTP, HTTP, telnet, etc.). Después, cada vez que un cliente de la intranet intente contactar con un servidor de ese tipo, el cliente será puesto en contacto, en realidad, con el servidor proxy. A su vez, el servidor proxy, desempeñando el papel de cliente contactará con el servidor auténtico. A partir de ese momento, el servidor proxy desempeña el papel de intermediario entre el cliente real y el servidor real, reenviando los mensajes entre uno y otro. La primera ventaja de este tipo de disposición es que el servidor real no tiene manera de saber que el servidor proxy no es el verdadero cliente y, de hecho, ni siquiera llega a ser consciente de la existencia del cliente real. A su vez, el servidor real no tiene forma de obtener datos acerca de las características internas de la intranet. La segunda ventaja es que el servidor proxy está en posición de filtrar todos los mensajes enviados desde el servidor al cliente. Por ejemplo, un servidor proxy FTP podría comprobar todos los archivos entrantes en busca de virus conocidos para bloquear todos los archivos infectados.

Otra herramienta más para prevenir problemas en un entorno de red es el software de auditoría, que es similar al software de auditoría del que ya hemos hablado en la sección dedicada a la seguridad de los sistemas operativos (Sección 3.5). Con el software de auditoría de red, un administrador de sistemas puede detectar un incremento súbito en el tráfico de mensajes en distintas ubicaciones dentro de su ámbito de responsabilidad. Puede también monitorizar las actividades de los cortafuegos del sistema y analizar los patrones de las solicitudes realizadas por las computadoras individuales con el fin de detectar irregularidades. En la práctica, el software de auditoría es una de las principales herramientas que utilizan los administradores para identificar problemas, antes de que estos pasen a estar fuera de control.

Otro medio de defensa frente a las invasiones a través de las conexiones de red es el denominado **software antivirus**, que se utiliza para detectar y eliminar virus conocidos y otras infecciones (en la práctica, el software antivirus representa una amplia clase de productos software, cada uno de ellos diseñado para detectar y eliminar un tipo específico de infección. Por ejemplo, mientras que muchos productos se especializan en el control de virus, otros están especializados en la protección frente al software espía). Es importante que los usuarios de estos paquetes entiendan que, al igual que en el caso de los sistemas biológicos, continuamente están apareciendo en escena nuevas infecciones de computadora, las cuales requieren vacunas actualizadas. Por tanto, el software antivirus debe contar con un mantenimiento periódico, consistente en descargar las actualizaciones proporcionadas por el fabricante del software. Sin embargo, ni siquiera esto garantiza la seguridad de una computadora. Después de todo, cada nuevo virus que aparece debe primero infectar algunas computadoras antes de ser descubierto y antes de que se pueda crear una vacuna. Por tanto, los usuarios inteligentes nunca abren adjuntos de correo electrónico procedentes de fuentes desconocidas, ni tampoco descargan soft-

ware sin confirmar antes su fiabilidad, ni responden a los anuncios que aparecen en ventanas emergentes, ni dejan un PC conectado a Internet cuando esa conexión no es necesaria.

Cifrado

En algunos casos, el propósito de los ataques vandálicos a través de la red es provocar fallos en el sistema (como en el caso de los ataques por denegación de servicio), pero en otros casos el objetivo último es obtener acceso a la información. El método tradicional para proteger la información es controlar el acceso a la misma mediante el uso de contraseñas. Sin embargo, las contraseñas pueden ser averiguadas y no son muy útiles cuando los datos se transfieren a través de redes e interredes en las que los mensajes son reenviados por una serie de entidades desconocidas. En estos casos, puede emplearse algún mecanismo de cifrado, de manera que aunque los datos caigan en manos poco escrupulosas, la información codificada continúe siendo confidencial. Hoy día, muchas aplicaciones tradicionales de Internet han sido modificadas para incorporar técnicas de cifrado, generándose así las denominadas "versiones seguras" de esas aplicaciones. Como ejemplos podríamos citar **FTPS**, que es una versión segura de FTP y SSH, que ya hemos presentado en la Sección 4.2 como un sustituto seguro para telnet.

Otro ejemplo más sería la versión segura de HTTP, conocida como **HTTPS**, que es utilizada por la mayoría de las instituciones financieras para proporcionar a los clientes acceso seguro a sus cuentas a través de Internet. El corazón de HTTPS es el sistema de protocolos conocido como **SSL** (*Secure Sockets Layer*, Capa de sockets seguros), desarrollado originalmente por Net-scape para proporcionar enlaces de comunicación seguros entre clientes y servidores web. La mayoría de los exploradores indican el uso de SSL mostrando un pequeño icono de un candado en la pantalla de la computadora. (Algunos utilizan la presencia o ausencia de ese icono para indicar si se está empleando SSL; otros muestran el candado abierto o cerrado.)

Uno de los temas más fascinantes en el campo del cifrado es la **criptografía de clave pública**, que utiliza una serie de técnicas que permiten diseñar los sistemas de cifrado de tal manera que aunque sepamos cómo están cifrados los mensajes, resulta imposible descifrarlos. Esta característica es bastante antiintuitiva. Después de todo, la intuición nos sugiere que si una persona sabe

Pretty Good Privacy

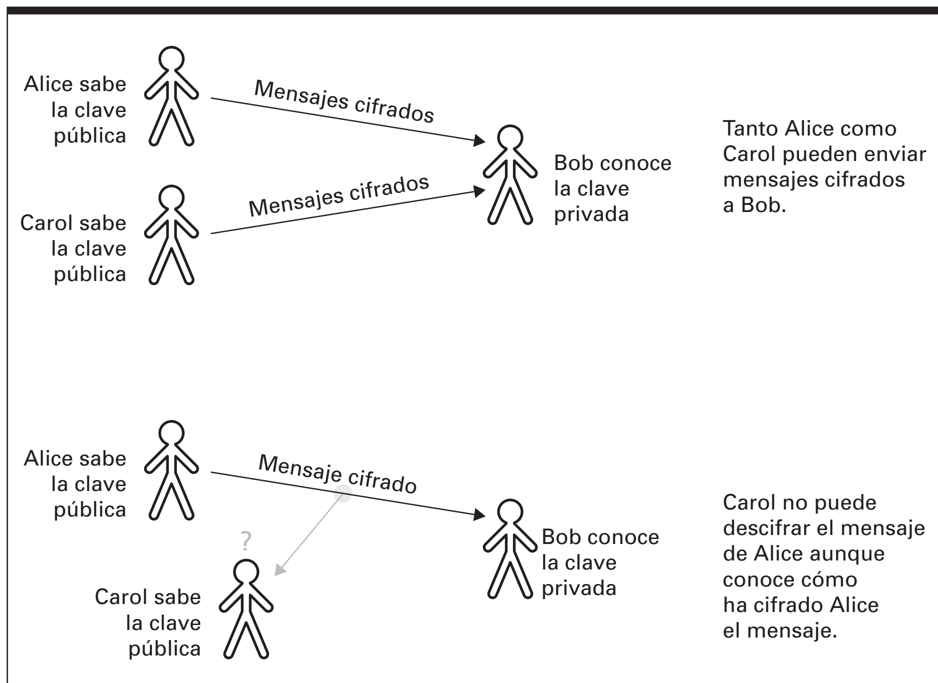
Quizá los sistemas de cifrado de clave pública más populares utilizados en Internet son los basados en el algoritmo RSA, que debe su nombre a sus inventores Ron Rivest, Adi Shamir y Len Adleman y que veremos en detalle al final del Capítulo 12. Las técnicas RSA (entre otras) se emplean en un conjunto de paquetes software fabricados por PGP Corporation. PGP son las siglas de Pretty Good Privacy. Estos paquetes son compatibles con la mayoría de las aplicaciones software de correo electrónico utilizadas en los PC y están disponibles de forma gratuita para uso personal y no comercial en la dirección http:// www.pgp.com. Con el software PGP, una persona puede generar una pareja de claves pública y privada, cifrar mensajes con claves públicas y descifrarlos mediante claves privadas.

cómo se cifran los mensajes, entonces esa persona debería ser capaz de invertir el proceso de cifrado, para así descifrar los mensajes. Pero los sistemas de cifrado de clave pública desafían esta lógica intuitiva.

Un sistema de cifrado de clave pública implica el uso de dos valores denominados **claves**. Una de las claves, la clave pública, se utiliza para cifrar los mensajes; la otra clave, la **clave privada**, es necesaria para descifrar los mensajes. Para utilizar el sistema, primero se distribuye la clave pública a aquellos que puedan necesitar enviar mensajes a un cierto destino concreto. La clave privada se mantiene almacenada de manera confidencial en dicho destino. Entonces, el creador de un mensaje puede cifrarlo utilizando esa clave pública y enviar el mensaje a su destino, con la garantía de que su contenido estará seguro, incluso a pesar de que vaya a ser manejado por una serie de intermediarios que también conocen la clave pública. De hecho, el único que puede descifrar el mensaje es el interlocutor situado en la máquina de destino del mensaje, que es quien posee la clave privada. Por tanto, si Bob crea un sistema de cifrado de clave pública y proporciona esa clave pública tanto a Alice como a Carol, ambas podrán cifrar mensajes dirigidos a Bob, pero ninguna de las dos podrá espiar las comunicaciones de la otra. De hecho, si Carol intercepta un mensaje de Alice, no podrá descifrarlo aun cuando sepa cómo lo ha cifrado (Figura 4.16).

Existen, por supuesto, una serie de sutiles problemas en los sistemas de clave pública. Uno de ellos consiste en garantizar que la clave pública que se esté utilizando sea, verdaderamente, la clave adecuada para el interlocutor de destino. Por ejemplo, si nos estamos comunicando con nuestro banco, tenemos que asegurarnos de que la clave pública que estemos empleando para el cifrado sea la del banco y no la de un impostor. Si un impostor se hiciera pasar por el banco

Figura 4.16 Cifrado de clave pública.



(un ejemplo de suplantación) y nos diera su clave pública, los mensajes que cifráramos y enviáramos al "banco" podrían ser descifrados por el impostor y no por nuestro banco. Por tanto, la tarea de asociar las claves públicas con los interlocutores correctos tiene una gran importancia.

Una técnica para resolver este problema consiste en establecer una serie de sitios Internet de confianza, denominados **autoridades de certificación**, cuyo trabajo consiste en mantener listas actualizadas de interlocutores, junto con sus claves públicas. Estas autoridades, actuando como servidores, proporcionan entonces información fiable sobre claves públicas a sus clientes, mediante una serie de paquetes conocidos con el nombre de certificados. Un **certificado** es un paquete que contiene el nombre de un interlocutor y su clave pública. En Internet, hay disponibles muchas autoridades de certificación comerciales, aunque también es bastante común que las organizaciones de gran tamaño mantengan sus propias autoridades de certificación, con el fin de tener un control más estrecho sobre la seguridad de las comunicaciones de dicha organización.

Finalmente, es necesario decir unas palabras sobre el papel que los sistemas de cifrado de clave pública desempeñan a la hora de resolver problemas de **autentificación**, que es el proceso de asegurarse de que el autor de un mensaje es, de hecho, quien dice ser. El punto crítico en esa tarea de autenticación es que, en algunos sistemas de cifrado de clave pública, los papeles de las claves de cifrado y de descifrado pueden invertirse. Es decir, puede cifrarse texto con la clave privada y, como solo hay un único interlocutor con acceso a dicha clave, cualquier texto que haya sido cifrado de esa manera tendrá su origen en dicho interlocutor. De este modo, el poseedor de la clave privada puede generar un patrón de bits, denominado **firma digital**, que solo ese interlocutor sabe cómo generar. Asociando dicha firma a un mensaje, el emisor puede marcar dicho mensaje como auténtico. Una firma digital puede ser tan simple como la versión cifrada del propio mensaje; en ese caso, lo único que tiene que hacer el emisor es cifrar el mensaje que quiere transmitir utilizando su propia clave privada (la clave que normalmente se utilizaría para el descifrado). Cuando se recibe el mensaje en el destino, el receptor usa la clave pública del emisor para descifrar la firma. El mensaje así revelado tendrá que ser necesariamente auténtico porque solo el poseedor de la clave privada podría haber generado esa versión cifrada.

Enfoques legales de la seguridad de red

Otra forma de mejorar la seguridad de los sistemas de redes de computadoras consiste en aplicar remedios legales. Sin embargo, existen dos obstáculos en este sentido. El primero es que el declarar como ilegal una cierta actividad no impide necesariamente que alguien la realice. Lo único que proporciona es la posibilidad de recurrir a la Justicia. El segundo obstáculo es que la naturaleza internacional de la comunicación por red hace que ese recurso a la Justicia sea muy difícil, porque lo que es ilegal en un país podría ser legal en otro. En último término, el objetivo de mejorar la seguridad de las redes mediante la protección legal es un proyecto de carácter internacional, que debe por tanto ser gestionado por organismos legales internacionales, un actor potencial en este campo sería el Tribunal Penal Internacional de La Haya.

Una vez hechas estas aclaraciones, es necesario admitir, de todos modos, que aunque la protección legal no es perfecta, sí que sigue teniendo una tremenda influencia, por lo que conviene analizar algunos de los pasos que se están dando para resolver determinados conflictos en el campo de las redes de comunicaciones. Para este propósito vamos a utilizar algunos ejemplos relativos a las leyes federales vigentes en Estados Unidos, aunque se podrían mencionar ejemplos similares derivados de las normas aplicables en otros lugares, como por ejemplo en la Unión Europea.

Comencemos con la proliferación de software malicioso. En Estados Unidos, este problema está contemplado en la Ley de fraudes y abusos informáticos (***Computer Fraud and Abuse Act***), que fue aprobada en 1984, aunque después ha sido modificada varias veces. Es esta ley la que ha permitido ejercer la acusación en la mayoría de los casos relacionados con la introducción de gusanos y virus. En pocas palabras, la ley requiere que se demuestre que el acusado provocó conscientemente la transmisión de un programa o de un conjunto de datos que causaron daños intencionadamente.

Esta ley también cubre los casos relativos al robo de información. En particular, la ley declara ilegal la obtención de cualquier cosa que tenga un determinado valor mediante el acceso no autorizado a una computadora. Los tribunales han tendido a realizar una interpretación amplia de la frase “cualquier cosa que tenga un determinado valor”, de modo que esa ley se ha aplicado a más cuestiones además de al robo de información. Por ejemplo, los tribunales han dictaminado que el simple uso de una computadora puede constituir “cualquier cosa que tenga un determinado valor”.

El derecho a la intimidad es otro de los problemas de las redes a los que se enfrenta la comunidad legal y además es uno de los más controvertidos. Entre las cuestiones planteadas se incluyen el derecho de un empresario a monitorizar las comunicaciones de sus empleados y el grado con el que el proveedor de un servicio Internet está autorizado a acceder a la información comunicada por sus clientes. Ambas cuestiones han provocado un considerable debate. En Estados Unidos, muchos de estos problemas están contemplados en la Ley de intimidad de las comunicaciones electrónicas (ECPA, ***Electronic Communication Privacy Act***) de 1986, que tiene su origen en la legislación destinada a controlar las escuchas telefónicas. Aunque la ley es muy larga, su objetivo está resumido en unos pocos párrafos. En particular, dicha ley establece que

> Excepto cuando se diga específicamente otra cosa en este capítulo, cualquier persona que intencionadamente intercepte, trate de interceptar o haga que cualquier otra persona intercepte o trate de interceptar cualquier comunicación por cable, oral o electrónica... será castigada tal como se indica en la subsección (4) o será procesada según lo dispuesto en la subsección (5).

y

> ... cualquier persona o entidad que proporcione un servicio de comunicación electrónico al público no podrá divulgar intencionadamente el contenido de cualquier comunicación... realizada a través de dicho servicio a ninguna persona o entidad distintas del destinatario o pretendido receptor de dicha comunicación, o un agente de tal destinatario o pretendido receptor.

En resumen, la ECPA confirma el derecho de las personas a la intimidad de sus comunicaciones, es ilegal que el proveedor de un servicio Internet divulgue

información acerca de las comunicaciones de sus clientes y es ilegal que las personas no autorizadas espíen las comunicaciones de otras. Pero la ECPA deja mucho margen para el debate. Por ejemplo, la cuestión relativa al derecho de un empresario a monitorizar las comunicaciones de sus empleados se convierte en una cuestión de autorización, que los tribunales han tendido a conceder a los empresarios cuando esas comunicaciones se realizan utilizando los equipos del empresario.

Además, la ley proporciona a algunos organismos gubernamentales la autoridad de monitorizar las comunicaciones electrónicas con ciertas restricciones. Estas normas han dado lugar a un acalorado debate. Por ejemplo, en 2000, el FBI reveló la existencia de un sistema, denominado Carnivore, que proporciona información acerca de las comunicaciones de todos los abonados de un proveedor de servicios Internet en lugar de solo acerca de los abonados para los que los tribunales hayan autorizado la monitorización de las conversaciones. Y en 2001, en respuesta al ataque terrorista contra el World Trade Center, el congreso de Estados Unidos aprobó la controvertida ley USA PATRIOT (***Uniting and Strengthening America by Providing Appropriate Tools Required to Intercept and Obstruct Terrorism***), que modificaba las restricciones que limitaban la operación de los organismos gubernamentales.

Además de las controversias legales y éticas suscitadas por estos desarrollos, la concesión de derechos de monitorización hacen que surjan algunos problemas técnicos que son más pertinentes para nuestro análisis. Uno de ellos es que para proporcionar estas capacidades, es necesario construir y programar un sistema de comunicaciones para poder monitorizar las comunicaciones deseadas. Establecer dichas capacidades era precisamente el objetivo de la Ley de asistencia de comunicaciones para actividades policiales (CALEA, ***Communications Assistance for Law Enforcement Act***). Esta ley obliga a las operadoras de telecomunicaciones a modificar sus equipos para que admitan las conexiones de monitorización empleadas por las fuerzas de seguridad, un requisito que ha resultado muy complejo y costoso de satisfacer.

Otra cuestión controvertida implica la colisión entre el derecho del gobierno a monitorizar las comunicaciones y el derecho de las personas a emplear técnicas de cifrado. Si los mensajes que se están monitorizando están bien cifrados, la monitorización de las comunicaciones tiene un valor limitado para las fuerzas de seguridad. Los gobiernos de Estados Unidos, Canadá y Europa están considerando la adopción de sistemas que obligarían a registrar ante una determinada autoridad las claves de cifrado, pero esas pretensiones están siendo contestadas por las grandes empresas. Después de todo, la existencia del espionaje industrial hace que resulte perfectamente comprensible que la exigencia de registrar las claves de cifrado sea rechazada por muchas empresas y ciudadanos. ¿Qué grado de seguridad tendría ese sistema de registro?

Por último, como ejemplo de lo amplio que es el rango de cuestiones legales relacionadas con Internet, podemos citar la Ley anticiberocupas de protección del consumidor (***Anticybersquatting Consumer Protection Act***) de 1999, que está diseñada para proteger a las organizaciones de aquellos impostores que pudieran tratar de establecer nombres de dominios parecidos (los que realizan esas prácticas se llaman ciberocupas). La ley prohíbe el uso de nombres de dominio que sean idénticos a una marca registrada o que puedan inducir a confusión. Uno de los efectos es que aunque la ley no prohíbe la especulación

con nombres de dominios (el proceso de registrar nombres de dominio potencialmente deseables y posteriormente vender los derechos sobre dichos nombres), limita dicha práctica a nombres de dominio genéricos. Por tanto, un especulador de nombres de dominio podría legalmente registrar un nombre genérico como `CochesUsadosBaratos.com` pero no podría reclamar derechos sobre el nombre `CochesUsadosDonPepe.com` si la marca Don Pepe ya está siendo usada en el negocio de los coches usados. Tales distinciones suelen ser el tema de debate en las demandas basadas en la Ley de ciberocupas de protección al consumidor.

Cuestiones y ejercicios

1. ¿Qué es el ***phishing***? ¿Cómo se puede proteger a las computadoras frente a esa práctica?
2. ¿Qué diferencia hay entre los tipos de cortafuegos que pueden instalarse en la pasarela de un dominio y los que pueden instalarse en una máquina individual de dicho dominio?
3. Técnicamente, el término ***datos*** hace referencia a las representaciones de la información, mientras que ***información*** hace referencia al significado subyacente. ¿Qué es lo que protege el uso de contraseñas, los datos o la información? ¿Qué es lo que protegen las técnicas de cifrado, los datos o la información?
4. ¿Qué ventaja presenta el cifrado de clave pública sobre las técnicas de cifrado más tradicionales?
5. ¿Cuáles son los problemas asociados con los intentos de protegerse legalmente frente a los problemas de seguridad de las redes?

Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. ¿Cuáles son los componentes de una red?
2. ¿Qué es la topología de red?
3. ¿Cuáles son las características de una red?
4. Describa las ventajas de un sistema distribuido de computación.
5. ¿Cuál es la diferencia entre una red abierta y una red cerrada?
6. ¿Qué protocolo de acceso múltiple es aplicable a una red inalámbrica?
7. Describa los pasos que sigue una máquina que desea transmitir un mensaje en una red en la que se esté empleando el protocolo CSMA/CD.
8. ¿Qué es el problema del terminal oculto? Describa una técnica para resolverlo.
9. ¿En qué se diferencia una dirección MAC de una dirección IP?
10. ¿Cuáles son los distintos procesos que tienen lugar en una comunicación entre procesos?
11. ¿Cuál es la diferencia entre una intranet y una interred?
12. ¿Cuáles son los distintos esquemas de direccionamiento utilizados en las redes?

13. Inicialmente se pensaba que el uso de direcciones de 32 bits proporcionaría espacio suficiente para futuras expansiones, pero están resultando que aquella conjetura no es correcta. IPv6 utiliza direccionamiento de 128 bits. ¿Cree que el número de bits será suficiente? Justifique su respuesta. (Por ejemplo, puede comparar el número de direcciones posibles con el número de habitantes de la Tierra.)

14. Codifique cada uno de los siguientes patrones de bits utilizando notación decimal con puntos.
a. 000001010001001000100011
b. 1000000000100000
c. 0011000000011000

15. ¿Qué patrón de bits está representado por cada uno de los siguientes patrones en notación decimal con puntos?
a. 0.0
b. 26.19.1
c. 8.12.20.13

16. Suponga que nos indican que la dirección de un sistema terminal en Internet es 154.148.46.120. ¿De qué clase de dirección IP se trata?

17. ¿Qué es una búsqueda DNS?

18. Si la dirección Internet en formato mnemónico de una computadora es
`batman.batcave.metropolis.gov`
¿qué podemos deducir acerca de la estructura del dominio en el que está la máquina?

19. Explique los componentes de la dirección de correo `john@yahoo.co.in`.

20. En el contexto de VoIP, ¿cuál es la diferencia entre un adaptador telefónico analógico y teléfono integrado?

21. ¿Cuáles son los números de puerto a través de los cuales un cliente FTP se comunica con un servidor FTP?

22. ¿Cuál es la diferencia entre N-unidifusión y multidifusión?

23. Defina cada uno de los siguientes términos:
a. CGI.
b. FTP.
c. ISP.

24. Defina cada uno de los siguientes términos:
a. Hipertexto.
b. HTML.
c. Explorador.

25. Muchos usuarios de Internet emplean de forma intercambiable los términos ***Internet*** y ***World Wide Web***. ¿A qué hace referencia cada uno de estos dos términos?

26. Al visualizar un documento web simple, pida a su explorador que muestre el código fuente del documento. A continuación, identifique la estructura básica del documento. En particular, identifique la cabecera y el cuerpo y enumere algunas de las instrucciones de cada una de esas dos partes del documento.

27. Enumere cinco etiquetas HTML y describa su significado.

28. Modifique el siguiente documento HTML de modo que el texto “Topología de malla” esté vinculado al documento cuya dirección URL es `http://netcom.org/Topologia/Topologia _Malla.html`.

```
<html>
<head>
<title>Primera página web</title>
</head>
<body>
Topología de malla
<img src> = “Topologia_Malla.jpg”>
</body>
</html>
```

29. Identifique los errores existentes en el siguiente documento HTML y corríjalo.

```
<html>
<title>Primera página web</title>
<head>
</head>
<body>
<h7>Fundamentos de redes</h7>
<img src = “Topologia_Malla.jpg”>
</html>
</body>
```

30. Utilizando el estilo XML informal presentado en el texto, diseñe un lenguaje de

composición para representar expresiones algebraicas simples en forma de archivos de texto.

31. Utilizando el estilo XML informal presentado en el texto, diseñe un conjunto de etiquetas que un procesador de textos pudiera utilizar para marcar el texto subyacente. Por ejemplo, ¿cómo indicaría un procesador de textos qué palabras deben estar en negrita, cursiva, subrayadas, etc.?

32. Utilizando el estilo XML informal presentado en el texto, diseñe un conjunto de etiquetas que puedan emplearse para componer reseñas de películas, de acuerdo con la forma en que los elementos de texto aparecerán en una página impresa. Después diseñe un conjunto de etiquetas que puedan utilizarse para componer esas reseñas de acuerdo con el significado de los elementos del texto.

33. Utilizando el estilo XML informal presentado en el texto, diseñe un conjunto de etiquetas que pudieran emplearse para componer artículos sobre eventos deportivos de acuerdo con la forma en que los elementos de texto deban aparecer en una página impresa. Después diseñe un conjunto de etiquetas que puedan usarse para componer los artículos de acuerdo con el significado de los elementos del texto.

34. Identifique los componentes de la siguiente dirección URL y describa el significado de cada uno de ellos. `http://lifeforms.com/animals/moviestars/kermit.html`

35. Identifique los componentes de las siguientes direcciones URL abreviadas.

a. `http://www.farmtools.org/windmills.html`
b. `http://castles.org/`
c. `www.coolstuff.com`

36. ¿Cómo variaría la acción de un explorador si le pidiéramos "localizar el documento" situado en la dirección URL
`http://stargazer.universe.org`
con respecto a la dirección
`https://stargazer.universe.org?`

37. Proporcione dos ejemplos de actividades del lado del cliente en la Web. Proporcione dos ejemplos de actividades del lado del servidor en la Web.

***38.** ¿Cuáles son las capas del modelo de referencia de red ISO/OSI?

***39.** En una red basada en la topología de bus, el bus es un recurso no compartible por el que las máquinas tienen que competir con el fin de transmitir mensajes. ¿Cómo se controla el interbloqueo (véase la Sección opcional 3.4) en este contexto?

***40.** Cite las cuatro capas de la jerarquía del software de Internet e identifique una tarea realizada por cada una de las capas.

***41.** ¿Qué es HTTPS?

***42.** Cuando una aplicación pide a la capa de transporte que utilice TCP para transmitir un mensaje, ¿qué mensajes adicionales serán enviados por la capa de transporte para poder satisfacer la solicitud de la capa de aplicación?

***43.** ¿En qué sentido podría considerarse que TCP es un mejor protocolo que UDP para implementar la capa de transporte? ¿En qué sentido podría considerarse que UDP es mejor que TCP?

***44.** ¿Qué queremos decir cuando decimos que UDP es un protocolo no orientado a conexión?

***45.** ¿En qué capa de la jerarquía de protocolos TCP/IP podríamos colocar un cortafuegos para filtrar el tráfico entrante en función de los siguientes elementos:

a. Contenido del mensaje.
b. Dirección de origen.
c. Tipo de aplicación.

46. Suponga que quiere establecer un cortafuegos para filtrar los mensajes de correo electrónico que contienen ciertos términos y frases. ¿Dónde colocaría este cortafuegos en la pasarela del dominio o en el servidor de correo del dominio? Explique su respuesta.

47. ¿Qué es un cortafuegos y cuáles son las ventajas que proporciona?

48. Resuma los principios del cifrado de clave pública.

49. ¿Qué son los delitos informáticos?

50. ¿En qué sentido limita la naturaleza global de Internet las posibles soluciones legales a los problemas de Internet?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. La capacidad de conectar computadoras a través de redes ha popularizado el concepto de teletrabajo. ¿Podría indicar los pros y los contras de esta tendencia? ¿Afectará al consumo de recursos naturales? ¿Fortalecerá los lazos familiares? ¿Reducirá el "politiqueo" en las empresas? ¿Las personas que trabajen en casa tendrán las mismas oportunidades de promoción profesional que las que acuden a la oficina? ¿Se debilitarán los lazos comunitarios? ¿El menor contacto personal con los colegas tendrá un efecto positivo o negativo?
2. La compra de productos a través de Internet se está convirtiendo en una alternativa a la compra "presencial". ¿Qué efecto tendrá esa modificación de los hábitos de compra en las comunidades? ¿Y sobre los grandes almacenes? ¿Y qué sucede con los pequeños comercios, como las boutiques y las librerías, en los que a uno le gusta mirar sin necesariamente comprar algo? ¿Hasta qué punto es bueno o malo comprar al precio más bajo posible? ¿Existe alguna obligación moral de pagar más por un producto para apoyar a los comercios locales? ¿Es ético comparar productos en una tienda local y luego comprarlos a un precio más bajo a través de Internet? ¿Cuáles son las consecuencia a largo plazo de ese tipo de comportamiento?
3. ¿Hasta qué punto debe un gobierno controlar el acceso de sus ciudadanos a Internet (o a cualquier red de carácter internacional)? ¿Qué sucede con los problemas relativos a la seguridad nacional? ¿Podría citar algunos problemas de seguridad que pudieran surgir?
4. Los tablones de anuncios electrónicos permiten a los usuarios de redes publicar mensajes (a menudo de forma anónima) y leer los mensajes que otros publican. ¿Debemos considerar al administrador de uno de esos tablones de anuncios electrónicos responsable del contenido publicado por sus usuarios? ¿Debemos considerar a una empresa telefónica responsable del contenido de las conversaciones telefónicas que se mantienen a través de su red? ¿Debemos considerar al propietario de un supermercado responsable del contenido de un tablón de anuncios ubicado en su tienda?
5. ¿Debe monitorizarse el uso de Internet? ¿Debe regularse? En caso afirmativo, ¿quién debe encargarse de la regulación y hasta qué grado debe llegar esa regulación?

6. ¿Cuánto tiempo invierte utilizando Internet? ¿Considera que es un tiempo bien invertido? ¿Ha hecho el acceso a Internet que modifique sus actividades sociales? ¿Le resulta más fácil hablar con la gente a través de Internet que en persona?

7. Cuando compramos un paquete software para una computadora personal, el desarrollador suele pedirnos que nos registremos, para poder notificarnos las actualizaciones futuras. Este proceso de registro se hace cada vez más frecuentemente a través de Internet. Normalmente, nos piden que indiquemos nuestro nombre, dirección y tal vez la manera en que nos hemos enterado de la existencia del producto, y luego el software se encarga de transferir automáticamente estos datos al desarrollador. ¿Qué cuestiones éticas podrían suscitarse si el desarrollador diseñara el software de registro de modo que le enviara información adicional durante el proceso?; por ejemplo, el software podría analizar el contenido de nuestro sistema e informar de otros paquetes software que encuentre.

8. Cuando visitamos un sitio web, el sitio puede tener la capacidad de almacenar unos datos, denominados ***cookies***, en nuestra computadora para indicar que hemos visitado ese sitio. Después, esas ***cookies*** pueden emplearse para identificar a los visitantes que vuelven al sitio y para registrar sus actividades previas, de modo que las futuras visitas al sitio puedan ser gestionadas de manera más eficiente. Las ***cookies*** almacenadas en la computadora proporcionan también un registro de los sitios que hemos visitado. ¿Deberían tener los sitios web la capacidad de almacenar ***cookies*** en nuestra computadora? ¿Debería permitirse a un sitio web almacenar ***cookies*** en nuestra computadora sin nuestro consentimiento? ¿Qué ventajas potenciales tienen las ***cookies***? ¿Qué problemas podrían surgir a causa del uso de ***cookies***?

9. Si se obliga a las grandes corporaciones a registrar sus claves de cifrado ante un organismo gubernamental, ¿estarán seguras esas claves?

10. En general, las normas de etiqueta nos dicen que debemos evitar llamar a un amigo a su oficina para tratar de asuntos personales o sociales, como por ejemplo los preparativos para una escapada de fin de semana. Igualmente, la mayoría de nosotros tendría reparos en llamar a un cliente a su casa para describirle un nuevo producto. Por los mismos motivos, solemos enviar las invitaciones de boda a los domicilios de los invitados, mientras que los anuncios de presentaciones de carácter comercial los envíamos a la oficina del destinatario. ¿Es adecuado enviar un correo electrónico de carácter personal a un amigo a través del servidor de correo de la empresa de ese amigo?

11. Suponga que el propietario de un PC deja su máquina conectada a Internet, por lo que esta termina siendo utilizada por otra persona para implementar un ataque por denegación de servicio. ¿Hasta qué punto podría considerarse responsable al propietario del PC? ¿Depende su respuesta de si el propietario instaló los cortafuegos apropiados?

12. ¿Es ético que las empresas que fabrican caramelos o juguetes incluyan juegos en los sitios web de su empresa para entretener a los niños mientras que promocionan los productos de la empresa? ¿Y qué sucede si el juego está diseñado para recopilar información acerca de los niños? ¿Cuáles son las fronteras entre el entretenimiento, la publicidad y la explotación?

Lecturas adicionales

Antoniou, G. y F. van Harmelem. ***A Semantic Web Primer***. Cambridge, MA: MIT Press, 2004.

Bishop, M. ***Introduction to Computer Security***. Boston, MA: Addison-Wesley, 2005.

Comer, D. E. ***Computer Networks and Internets,*** 5ª ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Comer, D. E. ***Internetworking with TCP/IP,*** Vol. 1, 5ª ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Goldfarb, C. F. y P. Prescod. ***The XML Handbook,*** 5ª ed. Upper Saddle River, NJ: Prentice-Hall, 2004.

Halsal, F. ***Computer Networking and the Internet.*** Boston, MA: Addison-Wesley, 2005.

Harrington, J. L. ***Network Security: A Practical Approach.*** San Francisco: Morgan Kaufmann, 2005.

Kurose, J. F. y K. W. Ross. ***Computer Networking: A Top Down Approach Featuring the Internet,*** 4ª ed. Boston, MA: Addison-Wesley, 2008.

Peterson, L. L. y B. S. Davie. ***Computer Networks: A Systems Approach,*** 3ª ed. San Francisco: Morgan Kaufmann, 2003.

Rosenoer, J. ***CyberLaw, The Law of the Internet***. Nueva York: Springer, 1997.

Spinello, R. A. y H. T. Tavani. ***Readings in CyberEthics***. Sudbury, MA: Jones and Bartlett, 2001.

Stallings, W. ***Cryptography and Network Security,*** 4ª ed. Upper Saddle River, NJ: Prentice-Hall, 2006.

Stevens, W. R. ***TCP/IP Illustrated,*** Vol. 1. Boston, MA: Addison-Wesley, 1994.

CAPÍTULO 5

Algoritmos

En el capítulo de introducción vimos que el tema central de las Ciencias de la computación es el estudio de los algoritmos. Ahora es el momento de que nos centremos en este tema fundamental. Nuestro objetivo es explorar gran parte de esta materia fundamental como para poder entender y apreciar verdaderamente las Ciencias de la computación.

Hemos visto que antes de que una computadora pueda realizar una tarea, hay que proporcionarle un algoritmo que le diga de forma precisa qué hacer; en consecuencia, el estudio de los algoritmos es la piedra angular de las Ciencias de la computación. En este capítulo vamos a presentar muchos de los conceptos fundamentales de este estudio, incluyendo los temas sobre el descubrimiento y la representación de los algoritmos, así como los principales conceptos sobre control de algoritmos que son la iteración y la recursión. También iremos presentando algunos algoritmos bastante conocidos de búsqueda y de ordenación. Comenzaremos repasando el concepto de algoritmo.

5.1 Concepto de algoritmo

En el capítulo de introducción hemos definido de manera informal el concepto de algoritmo como un conjunto de pasos que define cómo llevar a cabo una tarea. En esta sección, vamos a examinar más en profundidad este concepto fundamental.

Un repaso informal

Ya nos hemos encontrado con una multitud de algoritmos en nuestro estudio. Hemos visto algoritmos para la conversión de representaciones numéricas de un formato a otro, para la detección y corrección de errores en los datos, para la compresión y descompresión de archivos de datos, para el control de la multiprogramación en un entorno multitarea, etc. Además, hemos visto que el ciclo de máquina seguido por un procesador no es otra cosa que el siguiente algoritmo simple:

```
Mientras que no se haya ejecutado la instrucción de
parada, continuar ejecutando los siguientes pasos:
  a. Captar una instrucción.
  b. Decodificar la instrucción.
  c. Ejecutar la instrucción.
```

Como ilustra el algoritmo de la Figura 0.1, en el que describíamos un truco de magia, los algoritmos no están restringidos a las actividades técnicas. De hecho, los algoritmos se emplean en actividades tan mundanas como por ejemplo pelar guisantes:

```
Obtener una cesta de guisantes sin pelar y un cuenco
vacío.
Mientras queden guisantes sin pelar en la cesta,
continuar ejecutando los siguientes pasos:
  a. Tomar un vaina de la cesta.
  b. Abrir la vaina.
  c. Echar los guisantes de la vaina en el cuenco.
  d. Tirar la vaina.
```

De hecho, muchos investigadores creen que todas las actividades de la mente humana, incluyendo la imaginación, la creatividad y la toma de decisiones son en la práctica el resultado de la ejecución de algún algoritmo, una conjetura de la que volveremos a hablar cuando abordemos el tema de la inteligencia artificial (Capítulo 11).

Pero antes de continuar vamos a ver cuál es la definición formal de algoritmo.

Definición formal de un algoritmo

Los conceptos informales definidos de manera vaga resultan aceptables en la vida cotidiana y son bastante comunes, pero la ciencia debe estar basada en una terminología bien definida. Consideremos entonces la definición formal de algoritmo enunciada en la Figura 5.1.

Observe que la definición requiere que el conjunto de pasos que forman un algoritmo sea ordenado. Esto significa que los pasos de un algoritmo tienen que tener una estructura bien establecida en términos de su orden de ejecución. Sin embargo, esto no quiere decir que los pasos deban ejecutarse en una secuencia compuesta de un primer paso seguida de un segundo, y así sucesivamente. Algunos algoritmos, conocidos con el nombre de algoritmos paralelos, contienen más de una secuencia de pasos, cada una de ellas diseñada para ser ejecutada por distintos procesadores en una máquina multiprocesador. En tales casos, el algoritmo global no posee una única secuencia de pasos adaptada a ese escenario compuesto por un primer paso, un segundo paso, etc. En lugar de ello, la estructura del algoritmo es la de una serie de múltiples hilos de ejecución que se ramifican y reconectan a medida que los distintos procesadores realizan diferentes partes de la tarea global (volveremos sobre este concepto en el Capítulo 6). Otros ejemplos incluyen los algoritmos ejecutados por circuitos tales como el biestable del Capítulo 1, en el que cada puerta realiza un único paso del algoritmo global. En ese ejemplo, los pasos están ordenados según causa y efecto, a medida que la acción de cada puerta se propaga a través del circuito.

A continuación, consideremos el requisito de que un algoritmo debe estar compuesto por pasos ejecutables. Para entender esta condición, considere la sentencia:

```
Hacer una lista de todos los enteros positivos
```

lo que sería imposible de hacer, porque existe un número infinito de enteros positivos. Por tanto, cualquier conjunto de sentencias en el que estuviera incluida esta, no sería un algoritmo. Los expertos en Ciencias de la computación emplean el término ***efectivo*** para reflejar el concepto de ser ejecutable. Es decir, cuando afirmamos que un paso es efectivo, quiere decir que se puede llevar a la práctica.

Otro requisito impuesto por la definición de la Figura 5.1 es que los pasos de un algoritmo no sean ambiguos. Esto significa que durante la ejecución de un algoritmo, la información acerca del estado del proceso debe ser suficiente como para determinar de forma unívoca y completa las acciones requeridas por cada paso. En otras palabras, la ejecución de cada paso de un algoritmo no

Figura 5.1 Definición de algoritmo.

Un algoritmo es un conjunto ordenado de pasos ejecutables y no ambiguos, que definen un proceso finito con un fin determinado.

requiere de ninguna capacidad creativa. En lugar de ello, tan solo requiere la capacidad de seguir las instrucciones que se proporcionen. (En el Capítulo 12 veremos que hay una serie de "algoritmos", denominados algoritmos no deterministas, que no cumplen esta restricción y que constituyen en la actualidad uno de los temas fundamentales de investigación.)

La definición de la Figura 5.1 también requiere que un algoritmo defina un proceso finito, lo que quiere decir que la ejecución de un algoritmo debe tener un final. El origen de este requisito está en las Ciencias de la computación, en las que el objetivo es responder a preguntas tales como "¿Cuáles son los límites últimos de los algoritmos y de las máquinas?" Aquí, las Ciencias de la computación tratan de establecer una diferencia entre aquellos problemas cuyas respuestas pueden obtenerse algorítmicamente y aquellos otros cuyas respuestas caen fuera de las capacidades de los sistemas algorítmicos. En este contexto, lo que hacemos es trazar una línea entre aquellos procesos que terminan proporcionando una respuesta y aquellos que simplemente se limitan a continuar ejecutándose eternamente, sin llegar a generar un resultado.

Sin embargo, existen aplicaciones importantes para procesos que no terminen; entre esas aplicaciones podríamos incluir la monitorización de los signos vitales de un paciente hospitalario o el mantenimiento de la altitud de una aeronave durante el vuelo. Alguien podría argumentar que estas aplicaciones simplemente implican la repetición de algoritmos, cada uno de los cuales finaliza en algún momento y luego se repite automáticamente. Otras personas podrían responder diciendo que dichos argumentos son meros intentos de ajustarse a una definición formal excesivamente restrictiva. En cualquier caso, el resultado es que el término ***algoritmo*** se utiliza a menudo en aplicaciones prácticas o en contextos informales para hacer referencia a conjuntos de pasos que no necesariamente definen procesos que termine. Un ejemplo sería un algoritmo para calcular la división exacta (resto igual a 0) de 1 y 3. Dado que tiene un número infinito de decimales, el proceso no puede ser finito. Técnicamente hablando, dichos ejemplos representan un uso inadecuado del término algoritmo.

La naturaleza abstracta de los algoritmos

Es importante hacer hincapié en la diferencia entre un algoritmo y su representación, una distinción análoga a la que existe entre una historia y un libro. Una historia es de naturaleza abstracta o conceptual; un libro es una representación física de una historia. Si traducimos un libro a otro idioma o lo volvemos a reeditar con un formato distinto, lo único que cambia es la representación de la historia, pero la propia historia continuará siendo la misma.

De la misma forma, un algoritmo es abstracto y difiere de su representación. Un mismo algoritmo puede representarse de muchas formas distintas. Por ejemplo, el algoritmo para convertir medidas de temperatura de Celsius a Fahrenheit se representa normalmente mediante la fórmula algebraica:

$$F = (9/5)C + 32$$

Pero podría representarse mediante la sentencia:

> Multiplicar la medida de temperatura en Celsius por $9/5$
> y luego sumar 32 al producto

o incluso puede representarse en forma de circuito electrónico. En cada caso, el algoritmo subyacente es el mismo, solo cambia la representación.

La distinción entre un algoritmo y su representación presenta un problema cuando intentamos comunicar algoritmos. Un ejemplo común es el relativo al grado de detalle con el que hay que describir un algoritmo. Entre los meteorólogos, la sentencia "Convertir la medida Celsius a su equivalente Fahrenheit" es suficiente, pero un lego, que necesitaría una descripción más detallada, podría decir que la sentencia es ambigua. Sin embargo, el problema no se refiere al algoritmo subyacente, sino al hecho de que ese algoritmo no ha sido representado con el suficiente detalle como para que un lego lo ejecute. En la siguiente sección veremos cómo puede utilizarse el concepto de primitivas para eliminar los problemas de ambigüedad en la representación de un algoritmo.

Finalmente, y antes de dejar el tema de los algoritmos y sus representaciones, es necesario clarificar la diferencia entre otros conceptos relacionados: los programas y los procesos. Un programa es una representación de un algoritmo. (Aquí estamos usando el término ***algoritmo*** en su sentido menos formal, en el sentido de que muchos programas son representaciones de "algoritmos" que no terminan.) De hecho, dentro de la comunidad informática, el término ***programa*** normalmente hace referencia a la representación formal de un algoritmo diseñado para su ejecución en una computadora. En el Capítulo 3, hemos definido un ***proceso*** como la actividad de ejecutar un programa. Observe, sin embargo, que ejecutar un programa consiste en ejecutar el algoritmo representado por el programa, por lo que también podríamos definir un proceso como la actividad de ejecutar un algoritmo. Como conclusión, podemos decir que los programas, los algoritmos y los procesos son entidades diferentes, aunque relacionadas. Un programa es la representación de un algoritmo, mientras que un proceso es la actividad de ejecutar un algoritmo.

Cuestiones y ejercicios

1. Resuma las diferencias entre un proceso, un algoritmo y un programa.
2. Proporcione algunos ejemplos de algoritmos con los que esté familiarizado. ¿Son realmente algoritmos en el sentido preciso del término?
3. Identifique algunos puntos vagos en nuestra definición informal de algoritmo presentada en la Sección 0.1 del capítulo de introducción.
4. ¿En qué sentido no se ajustan a la definición técnica de algoritmo los pasos descritos por la siguiente lista de instrucciones?

   Paso 1. Sacar una moneda del bolsillo y ponerla sobre la mesa.

   Paso 2. Volver al Paso 1.

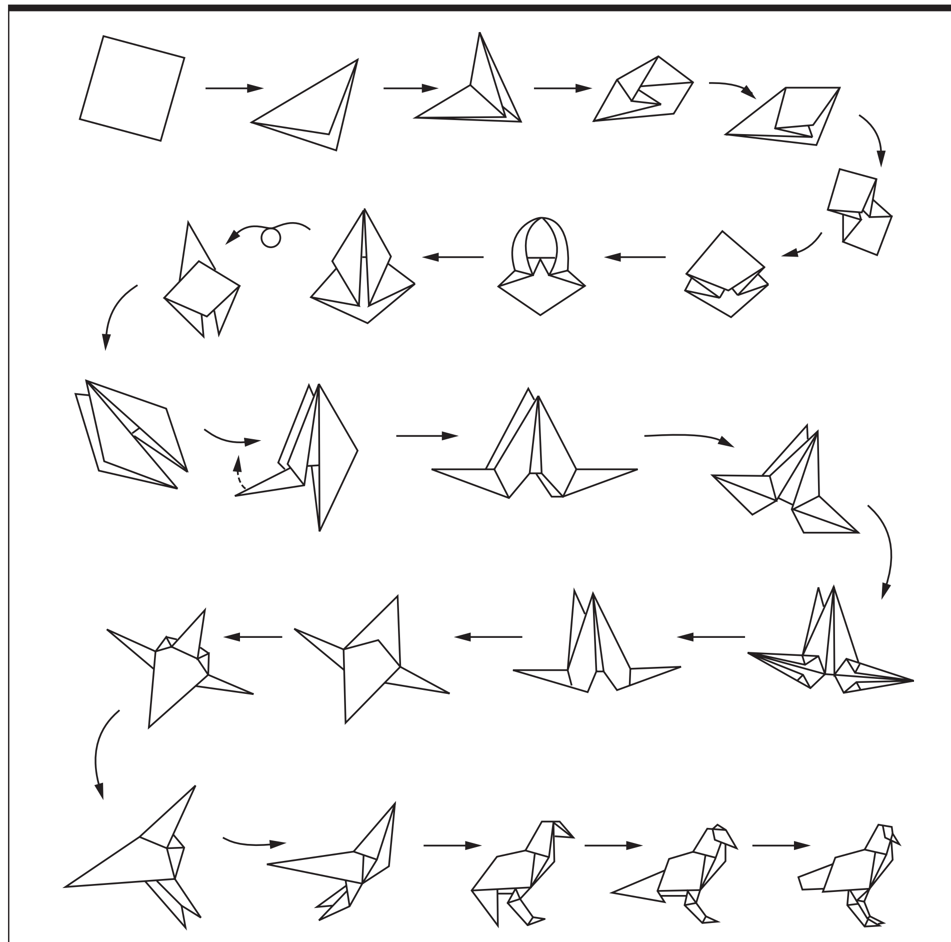
5.2 Representación de algoritmos

En esta sección vamos a considerar los temas relativos a la representación de un algoritmo. El objetivo es introducir los conceptos básicos de primitivas y pseudocódigo, así como establecer un sistema de representación para nuestro uso.

Primitivas

La representación de un algoritmo requiere algún tipo de lenguaje. En el caso de los seres humanos, este puede ser un lenguaje natural (inglés, español, ruso, japonés) o quizá un lenguaje gráfico, como el que se ilustra en la Figura 5.2, el cual describe un algoritmo para obtener la figura de un pájaro a partir de una hoja de papel cuadrada. A menudo, dichos canales naturales de comunicación conducen a que se produzcan malentendidos, en ocasiones porque la terminología utilizada tiene más de un significado (la frase "El azul me da pánico" podría significar que uno siente aversión hacia el color azul o que hay un objeto concreto de color azul que le causa miedo). También pueden surgir problemas debidos a malentendidos concernientes al nivel de detalle requerido. Pocos de los lectores serían capaces de construir un pájaro a partir de las instrucciones dadas en la Figura 5.2, mientras que un estudiante de origami tendría probablemente pocas dificultades. En resumen, pueden surgir problemas de comunicación cuando el lenguaje utilizado para representar un algoritmo no está definido de forma precisa o cuando no se proporciona la información con el suficiente detalle.

Figura 5.2 Construcción de un pájaro a partir de una hoja de papel cuadrada.



Las Ciencias de la computación tratan de resolver estos problemas estableciendo un conjunto bien definido de elementos fundamentales de construcción de software (*building block*) a partir de los cuales puedan construirse representaciones de algoritmos. Esos elementos se denominan **primitivas**. La asignación de definiciones precisas a estas primitivas elimina muchos problemas de ambigüedad y el exigir que los algoritmos se definan en términos de estas primitivas establece un nivel de detalle uniforme. Un conjunto de primitivas junto con una serie de reglas que indiquen cómo pueden combinarse esas primitivas para representar ideas más complejas constituye un **lenguaje de programación**.

Cada primitiva tiene su propia sintaxis y semántica. La sintaxis hace referencia a la representación simbólica de la primitiva; la semántica hace referencia al significado de la primitiva. La sintaxis de *aire* está compuesta por cuatro símbolos, mientras que la semántica establece que es una sustancia gaseosa que rodea a nuestro planeta. Como ejemplo, en la Figura 5.3 se muestran algunas de las primitivas utilizadas en origami.

Para obtener un conjunto de primitivas con el fin de utilizarlas para representar algoritmos destinados a su ejecución en una computadora, podemos fijarnos en cada una de las instrucciones que la máquina está diseñada para eje-

Figura 5.3 Primitivas de origami.

Sintaxis	Semántica
	Dar la vuelta al papel como en
Sombrear un lado del papel	Distingue entre diferentes lados del papel como en
	Representa un pliegue de tipo valle de forma que representa
	Representa un pliegue de tipo cima de forma que representa
	Doblar de modo que produce
	Doblar hacia dentro de modo que produce

cutar. Si expresamos un algoritmo con este nivel de detalle, dispondremos sin ninguna duda de un programa adecuado para su ejecución en esa máquina. Sin embargo, el expresar algoritmos de forma tan detallada resulta tedioso por lo que solemos emplear un conjunto de primitivas de "alto nivel", cada una de los cuales es una herramienta abstracta construida a partir de las primitivas de "bajo nivel" proporcionadas por el lenguaje máquina. El resultado es un lenguaje de programación formal, en el que los algoritmos pueden expresarse con un nivel conceptualmente mayor que en el lenguaje máquina. Hablaremos de esos lenguajes de programación en el siguiente capítulo.

Pseudocódigo

Por ahora, vamos a dejar a un lado la presentación de un lenguaje de programación formal para centrarnos en un sistema de notación menos formal y más intuitivo, conocido con el nombre de pseudocódigo. En general, un **pseudocódigo** es un sistema de notación en el que las ideas pueden expresarse informalmente durante el proceso de desarrollo del algoritmo.

Una forma de obtener un pseudocódigo es simplemente relajar las reglas del lenguaje formal en el que vaya a escribirse la versión final del algoritmo. Esta técnica se suele emplear cuando se conoce de antemano el lenguaje de programación previsto. En ese caso, el pseudocódigo utilizado durante las primeras etapas del desarrollo del programa está compuesto por estructuras sintáctico-semánticas similares a las utilizadas en el lenguaje de programación objetivo, aunque menos formales.

Representación de algoritmos durante el diseño de los mismos

La tarea de diseñar un algoritmo complejo requiere que el diseñador controle numerosos conceptos interrelacionados, un requisito que puede exceder las capacidades de la mente humana. Por tanto, el diseñador de algoritmos complejos necesita una forma de almacenar y recuperar partes de un algoritmo en evolución a medida que su trabajo lo requiera.

Durante las décadas de 1950 y 1960, los diagramas de flujo (mediante los cuales se representaban los algoritmos utilizando formas geométricas conectadas mediante flechas) constituían la herramienta de diseño más avanzada. Sin embargo, los diagramas de flujo se convirtieron pronto en una maraña entrelazada de flechas que se cruzaban, lo que hacía que comprender la estructura del algoritmo subyacente resultara difícil. Por ello, el uso de diagramas de flujo como herramienta de diseño ha dejado paso a otras técnicas de representación. Un ejemplo es el pseudocódigo utilizado en este texto mediante el que se representan los algoritmos usando estructuras textuales bien definidas. Los diagramas de flujo siguen siendo útiles cuando el objetivo es la presentación del algoritmo más que su diseño. Por ejemplo, las Figuras 5.8 y 5.9 aplican la notación de los diagramas de flujo para ilustrar la estructura algorítmica representada por algunas sentencias de control muy populares.

La búsqueda de mejores notaciones de diseño sigue evolucionando hoy día. En el Capítulo 7 veremos que la tendencia actual consiste en el uso de técnicas gráficas como ayuda para el diseño global de sistemas software de gran tamaño, mientras que el pseudocódigo sigue siendo popular para el diseño de los subprogramas, de menor tamaño, que forman un sistema.

Sin embargo, nuestro objetivo es considerar las cuestiones de desarrollo y representación de algoritmos sin reducir nuestro análisis a un lenguaje de programación concreto. Por tanto, nuestra técnica para la obtención de pseudocódigo consistirá en desarrollar una notación coherente y concisa para la representación de estructuras semánticas que se repiten. A su vez, esas estructuras serán las primitivas con las que trataremos de expresar futuras ideas.

Una de esas estructuras semánticas es el almacenamiento de un valor calculado. Por ejemplo, si hemos calculado la suma del saldo de nuestra cuenta corriente y de nuestra cuenta de ahorro, podríamos querer guardar el resultado para poder referirnos a él posteriormente. En estos casos, utiliza-ríamos la forma

nombre ← *expresión*

donde *nombre* es el nombre mediante el que haremos referencia al resultado y *expresión* describe el cálculo cuyo resultado queremos almacenar. Estas sentencias se leerían como "asignar a ***nombre*** el valor de ***expresión***" y nos referiremos a este tipo de sentencias como **sentencias de asignación**. Por ejemplo, la sentencia

```
SaldoTotal ← SaldoCC + SaldoAhorro
```

es una sentencia de asignación que asigna la suma de `SaldoCC` y `SaldoAhorro` al nombre `SaldoTotal`. Así, el término `SaldoTotal` puede utilizarse en sentencias futuras para referenciar ese valor.

Otra estructura semántica que aparece con frecuencia es la selección entre una de dos posibles actividades dependiendo de la verdad o falsedad de una cierta condición. Como ejemplos podríamos citar:

> Si el producto interior bruto se ha incrementado, comprar acciones; en caso contrario, vender acciones
> Comprar acciones si el producto interior bruto se ha incrementado y venderlas en caso contrario.
> Comprar o vender acciones dependiendo de si el producto interior bruto se ha incrementado o se ha reducido, respectivamente.

Cada una de estas frases podría reescribirse para adaptarse a la estructura

```
if (condición) then (actividad)
               else (actividad)
```

donde hemos utilizado las palabras clave **if**, **then** y **else** para presentar las diferentes subestructuras que componen la estructura principal y hemos usado paréntesis para delimitar las fronteras entre estas subestructuras. Adoptando esta estructura sintáctica para nuestro pseudocódigo, dispondremos de una manera uniforme con la que expresar esta estructura semántica común. Por tanto, aunque la sentencia

> Dependiendo de si el año es bisiesto o no, dividir el total entre 366 o 365, respectivamente.

tenga un estilo literario más creativo, optaremos por el estilo más simple

```
if (año es bisiesto)
   then (total diario ← total dividido entre 366)
   else (total diario ← total dividido entre 365)
```

También adoptaremos otra sintaxis abreviada

```
if (condición) then (actividad)
```

para aquellos casos que no impliquen una actividad alternativa. Con esta notación, la sentencia

En caso de que las ventas disminuyan, bajar el precio en un 5%.

se reduciría a

```
if (ventas disminuyen) then (bajar el precio un 5%)
```

Otra estructura semántica común es la ejecución repetida de una sentencia o de una secuencia de sentencias mientras continúe siendo cierta una determinada condición. Como ejemplos informales podríamos decir

Mientras que queden entradas por vender, seguir vendiendo entradas.

y

Hasta que se terminen de vender todas la entradas, continuar vendiéndolas.

Para estos casos, adoptaremos la estructura uniforme

```
while (condición) do (actividad)
```

en nuestro pseudocódigo. En resumen, una sentencia así indica que hay que comprobar una *condición* y, si es cierta, realizar la *actividad* y volver a comprobar de nuevo la *condición*. Sin embargo, si la *condición* resulta ser falsa, habrá que pasar a la sentencia que siga a la estructura `while`. Con esto, las dos sentencias anteriores se reducen a

```
while (queden entradas) do (vender una entrada)
```

A menudo el sangrado mejora la legibilidad de un programa. Por ejemplo, la sentencia

```
if (no llueve)
   then (if (temperatura = alta)
            then (ir a nadar)
            else (jugar al golf)
        )
   else (ver la televisión)
```

es más fácil de comprender que esta otra estructura equivalente

```
if (no llueve) then (if (temperatura = alta) then (ir a
nadar) else (jugar al golf)) else (ver la televisión)
```

Por tanto, vamos a utilizar el sangrado en nuestro pseudocódigo. (Observe que incluso podemos utilizar el sangrado para alinear un paréntesis de cierre directamente debajo de su paréntesis de apertura correspondiente, con el fin de simplificar el proceso de identificar el ámbito de las sentencias o frases.)

Queremos utilizar nuestro pseudocódigo para describir actividades que puedan emplearse como herramientas abstractas en otras aplicaciones. En las Ciencias de la computación se utilizan diversos términos para designar a esas unidades de programa; entre esos términos están subprograma, subrutina, procedimiento, módulo y función, cada uno de ellos con su propia variación de significado. En nuestro pseudocódigo nosotros adoptaremos para los procedi-

mientos el término `procedure` y lo usaremos para declarar el nombre con el que será conocida cada unidad de pseudocódigo. Es decir, una unidad de pseudocódigo comenzará con una sentencia de la forma

```
procedure nombre
```

donde *nombre* es el nombre concreto de la unidad. Después de esta primera sentencia incluiremos las sentencias que definen la acción llevada a cabo por esa unidad. Por ejemplo, la Figura 5.4 es una representación en pseudocódigo de un procedimiento denominado Saludos que imprime el mensaje "Hola" tres veces.

Cuando la tarea realizada por un procedimiento sea requerida en algún otro lugar de nuestro pseudocódigo, nos limitaremos a solicitarla utilizando su nombre. Por ejemplo, si llamamos a dos procedimientos `ProcesarPrestamo` y `RechazarSolicitud`, entonces podríamos solicitar sus servicios dentro de una estructura **if-then-else** escribiendo

```
if (...) then (Ejecutar el procedimiento ProcesarPrestamo)
         else (Ejecutar el procedimiento RechazarSolicitud)
```

que provocaría la ejecución del procedimiento `ProcesarPrestamo` si la condición verificada fuera cierta, mientras que en caso contrario, se ejecutaría el procedimiento `RechazarSolicitud`.

Si queremos poder utilizar los procedimientos en diferentes situaciones, es preciso diseñarlos de modo que sean lo más genéricos posible. Un procedimiento para ordenar listas de nombres debe diseñarse de forma que sea capaz de ordenar cualquier lista (no una lista concreta), así que debemos escribirlo de tal manera que la lista que haya que ordenar no esté especificada en el propio procedimiento. En lugar de ello, la lista debe ser referenciada mediante un nombre genérico dentro de la representación del procedimiento.

Nomenclatura de los elementos de un programa

En los lenguajes naturales, los elementos a menudo tienen nombres formados por varias palabras, como por ejemplo "coste de producción" u "hora estimada de llegada". La experiencia ha demostrado que utilizar estos nombres formados por varias palabras en la representación de un algoritmo puede complicar la descripción del mismo. Es mejor tener identificado a cada elemento mediante un único bloque de texto continuo. A lo largo de los años se han utilizado muchas técnicas para comprimir varias palabras en un sola unidad léxica con el fin de obtener nombres descriptivos para los elementos de un programa. Una de ellas consiste en utilizar el guión bajo para unir las palabras, lo que da lugar a nombres como `hora_estimada_llegada`. Otro método consiste en usar letras mayúsculas para ayudar al lector a entender un nombre formado por varias palabras. Por ejemplo, comenzando cada palabra con mayúscula para obtener nombres como `HoraEstimadaLlegada`. Esta técnica se denomina **Pascal casing**, porque fue popularizada por los usuarios del lenguaje de programación Pascal. Una variante de la técnica de Pascal es la notación **camel casing**, que es idéntica a la de Pascal excepto porque la primera letra del nombre se deja en minúscula, como por ejemplo en `horaEstimadaLlegada`. En este texto, vamos a emplear la técnica Pascal casing, pero la elección es por supuesto una cuestión de gustos.

Figura 5.4 El procedimiento Saludos en pseudocódigo.

```
procedure Saludos
Contador ← 3;
while (Contador > 0) do
   (imprimir el mensaje "Hola" y
    Contador ← Contador –1)
```

En nuestro pseudocódigo, adoptaremos el convenio de enumerar estos nombres genéricos (que se denominan **parámetros**) entre paréntesis, en la misma línea en la que identifiquemos el nombre del procedimiento. En particular, un procedimiento denominado `Ordenar`, diseñado para ordenar cualquier lista de nombres, comenzaría con la sentencia

```
procedure Ordenar (Lista)
```

Cuando más adelante en el procedimiento hiciera falta hacer una referencia a la lista que hay que ordenar, se utilizaría el nombre genérico `Lista`. A su vez, cuando hicieran falta los servicios de `Ordenar`, identificaríamos la lista que hay utilizar en lugar de `Lista` en el procedimiento `Ordenar`. Por tanto, escribiríamos algo así como

Aplicar el procedimiento Ordenar a la lista de miembros de la organización

y

Aplicar el procedimiento Ordenar a la lista de invitados a la boda

dependiendo de nuestras necesidades.

Recuerde que el propósito de nuestro pseudocódigo es proporcionar un medio de representar los algoritmos de una manera legible e informal. Queremos un sistema de notación que nos ayude a expresar nuestras ideas, no que nos esclavice obligándonos a ajustarnos a reglas formales rigurosas. Por tanto, nos sentiremos libres de expandir o modificar nuestro pseudocódigo cuando sea necesario. En particular, si las sentencias de otro conjunto de paréntesis incluyen otros conjuntos de sentencias entre paréntesis, puede llegar a resultar difícil emparejar visualmente los paréntesis de apertura y de cierre. En estos casos, muchas personas encuentran bastante útil poner a continuación de un paréntesis de cierre un breve comentario que explique qué sentencia o frase está siendo terminada. En particular, se puede incluir a continuación del paréntesis final de una sentencia `while` las palabras `end while`, obteniéndose una sentencia tal como

```
while (...) do
  (.
   .
   .
  )end while
```

o quizá

```
while (...) do
  (if (...)
```

```
        then (.
              .
              .
             )end if
    )end while
```

donde hemos indicado el final tanto de la sentencia `if` como de la `while`.

Lo importante es que estamos intentando expresar un algoritmo en un formato legible, por lo que introducimos ayudas visuales (sangrías, comentarios, etc.) en determinadas ocasiones con el fin de conseguir nuestro objetivo. Además, cuando nos encontremos con una estructura que aparece con frecuencia que todavía no haya sido incorporada en nuestro pseudocódigo, podemos decidir ampliar el pseudocódigo, por el procedimiento de adoptar una sintaxis coherente para representar ese nuevo concepto.

Cuestiones y ejercicios

1. Una primitiva en un cierto contexto puede resultar ser una composición de primitivas en otro. Por ejemplo, nuestra sentencia `while` es una primitiva en nuestro pseudocódigo, aunque en último término sería implementada como una composición de instrucciones en lenguaje máquina. Proporcione dos ejemplos de este fenómeno en un contexto no relacionado con las Ciencias de la computación.
2. ¿En qué sentido equivale la construcción de procedimientos a la construcción de primitivas?
3. El algoritmo de Euclides calcula el máximo común divisor de dos enteros positivos `X` e `Y` mediante el siguiente proceso:

   Mientras que ni el valor de `X` ni el de `Y` sean cero, continuar dividiendo el mayor de ambos valores entre el más pequeño y asignando a `X` e `Y` los valores del divisor y el resto, respectivamente. (El valor final de `X` es el máximo común divisor.)

   Exprese este algoritmo en nuestro pseudocódigo.
4. Describa un conjunto de primitivas utilizadas en algún campo distinto del de la programación de computadoras.

5.3 Descubrimiento de algoritmos

El desarrollo de un programa está compuesto por dos actividades: descubrir el algoritmo subyacente y representar dicho algoritmo en forma de programa. Hasta este punto, nos hemos fijado en las cuestiones relacionadas con la representación de algoritmos, sin plantearnos previamente cómo se descubren los algoritmos. Y sin embargo, el descubrimiento de algoritmos suele ser el paso más complicado dentro del proceso del desarrollo de software. Después de todo, descubrir un algoritmo para resolver un problema requiere encontrar un método de resolución de ese problema. Por tanto, comprender cómo se descubren los algoritmos equivale a comprender el proceso de resolución de problemas.

El arte de la resolución de problemas

Las técnicas de resolución de problemas y la necesidad de aprender más acerca de ellas no son características de las Ciencias de la computación, sino que se trata de temas que atañen a casi todos los campos del conocimiento. La estrecha asociación entre el proceso de descubrimiento de algoritmos y el de resolución general de problemas ha hecho que los expertos en computación unan sus fuerzas con los de otras disciplinas tratando de buscar mejores técnicas para la resolución de problemas. En último término, lo que uno quisiera es reducir el proceso de resolución de problemas a un algoritmo en sí mismo, pero se ha demostrado que esto es imposible (este es un resultado del material presentado en el Capítulo 12, donde demostraremos que existen problemas que no tienen solución algorítmica). Por tanto, la capacidad de resolver problemas sigue siendo más una habilidad artística que hay que desarrollar, que una ciencia precisa que haya que aprender.

Como prueba de la naturaleza artística y esquiva de la tarea de resolución de problemas, las siguientes fases (vagamente definidas) de resolución de problemas que presentó el matemático G. Polya en 1945 siguen siendo los principios básicos utilizados en la actualidad por muchos de aquellos que intentan enseñar capacidades para la resolución de problemas.

Fase 1. Comprender el problema.

Fase 2. Desarrollar un plan para resolver el problema.

Fase 3. Llevar a cabo el plan.

Fase 4. Evaluar la solución para comprobar su precisión y evaluar su potencial como herramienta para resolver otros problemas.

Traducidas al contexto del desarrollo de programas, estas fases serían

Fase 1. Comprender el problema.

Fase 2. Desarrollar una idea acerca de cómo podría resolver el problema un procedimiento algorítmico.

Fase 3. Formular el algoritmo y representarlo como un programa.

Fase 4. Evaluar el programa para comprobar su precisión y evaluar su potencial como herramienta para resolver otros problemas.

Habiendo presentado la lista de Polya, debemos recalcar que estas fases no son pasos que haya que seguir cuando se esté intentando resolver un problema, sino fases que serán completadas en algún momento indeterminado del proceso de solución. La palabra más importante es ***seguir***. Los problemas no se resuelven siguiendo nada. Para resolver un problema, es necesario tomar la iniciativa y ser uno mismo el que abra el camino. Si tratamos de afrontar la tarea de solucionar un problema con el estado mental descrito por la frase "Ahora he acabado con la fase 1, así que es el momento de pasar a la fase 2", es bastante poco probable que tengamos éxito. Sin embargo, si nos implicamos en el problema y llegamos a resolverlo, lo más probable es que podamos echar la vista atrás para analizar lo que hemos hecho y darnos cuenta de que hemos pasado por las distintas fases descritas por Polya.

Otra observación importante es que las fases de Polya no se completan necesariamente de forma secuencial. Aquellos que resuelven con éxito un problema, a menudo comienzan formulando estrategias para resolver el problema

(fase 2), antes incluso de haber llegado a comprender completamente el problema (fase 1). Entonces, si dichas estrategias fallan (durantes las fases 3 o 4), esa persona logra una mejor comprensión de los detalles del problema y, gracias a esa comprensión mejorada, puede volver atrás y formular otras estrategias supuestamente más adecuadas.

Recuerde que estamos hablando de cómo se resuelven los problemas, no de cómo nos gustaría que se resolvieran. Idealmente, desearíamos no tener que perder el tiempo en ese proceso de prueba y error que acabamos de describir. En el caso de desarrollo de grandes sistemas software, el descubrir en la fase 4 que se había comprendido mal un aspecto del problema puede representar un enorme derroche de recursos. Uno de los principales objetivos de los ingenieros de software (Capítulo 7), es precisamente evitar ese tipo de catástrofes. Por eso, esos ingenieros han insistido tradicionalmente en la necesidad de comprender completamente el problema antes de tratar de implementar una solución. Sin embargo, podría argumentarse que no es posible comprender verdaderamente un problema hasta que no se ha encontrado una solución. El simple hecho de que un problema no esté resuelto implica, en sí mismo, una falta de comprensión. Por tanto, insistir en que se comprenda completamente el problema antes de proponer ninguna solución es algo demasiado idealista.

Por ejemplo, considere el siguiente problema:

> Asignamos a la persona A la tarea de determinar la edades de los hijos de la persona B. B dice a A que el producto de las edades de los tres niños es igual a 36. Después de tener en cuenta este dato, A replica que hace falta algún dato adicional, por lo que B le dice a A cuál es la suma de las edades de sus hijos. De nuevo, A contesta que necesita otro dato, por lo que B le dice a A que su hijo mayor toca el piano. Después de escuchar esto, A le dice a B cuáles son las edades de sus tres hijos.
>
> ¿Qué años tienen cada uno de los niños?

A primera vista, el último dato parece no guardar ninguna relación con el problema, a pesar de lo cual es aparentemente ese dato el que permite a A determinar finalmente las edades de los tres niños. ¿Cómo puede ser esto? Vamos a abordar el problema formulando un plan de ataque y vamos a ceñirnos a él, aunque tenemos todavía muchas cuestiones sin responder acerca del problema. Nuestro plan consistirá en trazar los pasos descritos por el enunciado del problema mientras vamos anotando la información que la persona A tiene a su disposición a medida que va avanzando la historia.

El primer dato proporcionado a A es que al multiplicar las edades de los tres niños se obtiene el valor 36. Esto significa que la terna que representa las tres edades es una de las enumeradas en la Figura 5.5(a). La siguiente pista es

Figura 5.5

a. Ternas cuyo producto es 36		**b.** Sumas de las ternas del apartado (a)	
(1,1,36)	(1,6,6)	1 + 1 + 36 = 38	1 + 6 + 6 = 13
(1,2,18)	(2,2,9)	1 + 2 + 18 = 21	2 + 2 + 9 = 13
(1,3,12)	(2,3,6)	1 + 3 + 12 = 16	2 + 3 + 6 = 11
(1,4,9)	(3,3,4)	1 + 4 + 9 = 14	3 + 3 + 4 = 10

la suma de la terna deseada. No se nos dice cuál es esa suma, pero lo que sí se nos dice es que dicha información no es suficiente para que A identifique la terna correcta; por tanto, la terna deseada debe ser una cuya suma aparezca al menos dos veces en la tabla de la Figura 5.5(b). Pero las únicas ternas que aparecen en la Figura 5.5(b) con sumas idénticas son (1,6,6) y (2,2,9), ambas de las cuales nos dan como suma el valor 13. Esta es la información que tiene la persona A a su disposición en el momento en que se le proporciona el último dato. Es en este punto cuando finalmente comprendemos la importancia de ese último dato. No tiene nada que ver con tocar el piano; lo importante es el hecho de que hay un niño que es mayor que los otros dos, esto excluye la terna (1,6,6) y nos permite por tanto concluir que las edades de los niños son 2, 2 y 9.

En este caso, por tanto, podemos ver que hasta que no hemos intentado implementar nuestro plan para solucionar el problema (fase 3) no hemos comprendido completamente la naturaleza del mismo (fase 1). Si hubiéramos insistido en completar la fase 1 antes de continuar, probablemente nunca habríamos averiguado cuáles eran las edades de los niños. Este tipo de irregularidades en el proceso de resolución de problemas tiene una enorme importancia en lo que respecta a las dificultades que existen para desarrollar métodos sistemáticos de resolución de problemas.

Otra irregularidad es esa misteriosa inspiración que puede experimentar alguien que está intentando resolver un problema y que, habiendo trabajado en el mismo aparentemente sin ningún éxito, en algún momento posterior encuentra de repente la solución mientras está realizando otra tarea. Este fenómeno fue ya identificado por H. von Helmholtz en 1896 y fue analizado por el matemático Henri Poincaré en una conferencia pronunciada ante la Psychological Society en Paris. En esa conferencia, Poincaré describía sus propias experiencias en casos en los que había encontrado la solución a un problema en el que había estado trabajando después de haberlo dejado de lado y haber iniciado otros proyectos. Este fenómeno refleja un proceso en el que una parte subconsciente de nuestra mente parece continuar trabajando para, en caso de tener éxito, entregar esa solución a nuestra mente consciente. En la actualidad, ese periodo que media entre el trabajo consciente en un problema y la súbita inspiración se conoce con el nombre de periodo de incubación, y el tratar de comprender dicho fenómeno continúa siendo uno de los objetivos de las investigaciones actuales.

Abrirse camino

Hemos estado hablando de la resolución de problemas desde una perspectiva un tanto filosófica, mientras que evitábamos afrontar directamente la cuestión de cómo debemos intentar resolver un problema. Existen, por supuesto, numerosas técnicas de resolución de problemas, cada una de las cuales puede resultar adecuada en ciertos casos. Más adelante identificaremos algunas de ellas. Por ahora, vamos a señalar que parece existir un denominador común en todas estas técnicas, que podríamos enunciar de manera simple como "es necesario abrirse camino". Por ejemplo, consideremos el siguiente problema:

> Antes de participar en una carrera, A, B, C y D hacen las siguientes predicciones:
>
> A predice que B va a ganar.

B predice que D va a quedar en último lugar.
C predice que A va a quedar en tercera posición.
D predice que la predicción de A será correcta.

Finalizada la carrera, solo una de estas predicciones resulta ser correcta y se trata precisamente de la predicción hecha por el ganador. ¿En qué orden han terminado la carrera A, B, C y D?

Después de leer el problema y analizar los datos, se tarda poco en darse cuenta de que, como las predicciones de A y D son equivalentes y como solo una de las predicciones resultó ser correcta, tanto la predicción de A como la predicción de D tienen que ser falsas. Por tanto, ni A ni D han sido el ganador de la carrera. Llegados a este punto, ya hemos abierto brecha y la obtención de una solución completa al problema es cuestión simplemente de ir expandiendo nuestro conocimiento a partir de ahí. Si la predicción de A ha resultado ser falsa, entonces B tampoco ha sido el ganador, luego la única posibilidad que queda es que el ganador haya sido C. En consecuencia, C ha ganado la carrera y la predicción de C era cierta. En consecuencia, sabemos que A ha llegado en tercer lugar. Esto quiere decir que el orden de finalización fue CBAD o CDAB. Pero la primera de las dos soluciones es imposible porque la predicción de B tiene que ser falsa. Por tanto, el orden en el que los cuatro participantes finalizaron la carrera fue CDAB.

Por supuesto, el hecho de que nos digan que debemos abrir camino no nos dice nada acerca de cómo tenemos que abrirla. El proceso de abrir camino en el problema, al igual que el de comprender como expandir nuestro conocimiento a partir de ahí para obtener una solución completa del problema, requiere una labor creativa por parte de la persona que está intentando obtener la solución. Sin embargo, existen varias técnicas generales propuestas por Polya y otros para intentar ver cómo abrir camino. Una de esas técnicas consiste en trabajar con el problema en sentido inverso. Por ejemplo, si el problema consiste en encontrar una forma de generar una salida concreta a partir de una determinada entrada, podemos comenzar con dicha salida y tratar de retroceder hasta la entrada que nos hayan indicado. Esta técnica es típica de las personas que tratan de descubrir el algoritmo para la obtención de una figura de pájaro a partir de un papel cuadrado que hemos presentado en la sección anterior. Esas personas tienden a deshacer la figura, desdoblando el papel para ver cómo se ha construido.

Otra técnica general de resolución de problemas consiste en buscar un problema relacionado que sea más fácil de resolver o que ya haya sido resuelto con anterioridad y luego tratar de aplicar su solución al problema actual. Esta técnica es particularmente interesante en el contexto del desarrollo de programas. Generalmente, el desarrollo de un programa no es el proceso de resolver un caso particular de un problema sino más bien de encontrar un algoritmo general que puede usarse para resolver todos los casos de ese problema. Para ser más precisos, si se nos encarga la tarea de desarrollar un programa para ordenar alfabéticamente listas de nombres, nuestra tarea no sería ordenar una lista concreta, sino encontrar un algoritmo general que pudiera utilizarse para ordenar cualquier lista de nombres posible. Por tanto, aunque las sentencias

```
Intercambiar los nombres David y Alicia.
Mover el nombre Carol a la posición entre Alicia y David.
Mover el nombre Benito a la posición entre Alicia y Carol.
```

permitiría ordenar correctamente la lista de nombres David, Alicia, Carol y Benito, no constituye el algoritmo de propósito general que estamos buscando. Lo que necesitamos es un algoritmo que ordene tanto esta lista como otras que podamos encontrarnos. Esto no quiere decir que nuestra solución para ordenar una lista concreta sea completamente inútil en nuestra búsqueda de un algoritmo de propósito general. Por ejemplo, podemos abrir camino en el problema considerando algunos casos especiales, en un intento de encontrar principios generales que luego puedan utilizarse para desarrollar el algoritmo de propósito general deseado. En este caso, por tanto, la solución se obtiene mediante la técnica de resolver un conjunto de problemas relacionados.

Otra técnica más para tratar de abrir camino en el problema consiste en aplicar un **refinamiento sucesivo**, que es básicamente la técnica de no tratar de acometer de una sola vez la tarea completa (con todos sus detalles). En lugar de ello, el refinamiento sucesivo propone que tratemos primero de ver el problema que tenemos entre manos en términos de diversos subproblemas. La idea es que al descomponer el problema original en subproblemas, podemos ser capaces de aproximarnos a la solución global mediante una serie de pasos, cada uno de los cuales será más fácil de resolver que el problema original completo. A su vez, la técnica del refinamiento sucesivo propone que estos pasos se descompongan en otros pasos más pequeños y que esos otros, a su vez, se descompongan en pasos más pequeños todavía hasta que todo el problema se haya reducido a un conjunto de subproblemas fácilmente resolubles.

Desde este punto de vista, el refinamiento sucesivo es una **metodología arriba-abajo** en el sentido de que va progresando de lo más general a lo más específico. Por el contrario, una **metodología abajo-arriba** trata de progresar desde lo más específico hasta lo más general. Aunque ambas técnicas son muy diferentes en teoría, las dos suelen complementarse durante el proceso creativo de resolución de problemas. La descomposición propuesta por la metodología arriba-abajo de refinamiento sucesivo de un problema suele estar guiada por la intuición de la persona encargada de resolver el problema, que puede estar por su parte trabajando en modo abajo-arriba.

La metodología arriba-abajo de refinamiento sucesivo es esencialmente una herramienta organizativa cuyas propiedades para la resolución de problemas son consecuencia de dicha organización. Este tipo de técnica ha sido desde hace mucho tiempo una metodología importante dentro de la comunidad de procesamiento de datos, en la que el desarrollo de grandes sistemas software abarca una componente organizativa significativa. Pero, como veremos en el Capítulo 7, los grandes sistemas software se construyen cada vez más mediante la combinación de componentes prefabricados, una técnica que es inherentemente de tipo abajo-arriba. Por tanto, tanto la metodología arriba-abajo como la metodología abajo-arriba continúan siendo herramientas de gran importancia en las Ciencias de la computación.

Es muy importante tratar de mantener una perspectiva amplia. Y un ejemplo de dicha importancia es el hecho de que tratar de aplicar nociones preconcebidas y herramientas preseleccionadas a la tarea de resolución de problemas puede, en ocasiones, enmascarar la simplicidad de un determinado problema. El problemas de las edades de los niños que hemos presentado anteriormente en esta sección es un excelente ejemplo de este fenómeno. Los estudiantes de

álgebra siempre tratan de enfocar el problema mediante un sistema de ecuaciones, lo que es una técnica que conduce a un punto muerto y que a menudo lleva a la persona que está intentando solucionar el problema a creer, erróneamente, que la información proporcionada no es suficiente para resolver el problema.

Veamos el siguiente ejemplo:

> Al saltar a un bote de remos desde el muelle, se nos cae el sombrero al agua sin darnos cuenta. El río está fluyendo a 2,5 km por hora, por lo que nuestro sombrero comienza a flotar corriente abajo. Mientras tanto, nosotros comenzamos a remar corriente arriba a una velocidad de 4,75 km por hora respecto al agua. Después de 10 minutos, nos damos cuenta de que hemos perdido el sombrero, damos la vuelta al bote y comenzamos a perseguir al sombrero río abajo a la misma velocidad. ¿Cuánto tardaremos en alcanzar al sombrero?

La mayoría de los estudiantes de álgebra así como los entusiastas de las calculadoras, enfocan este problema determinando primero qué distancia río arriba habrá recorrido el bote en 10 minutos, así como la distancia río abajo que el sombrero habrá recorrido en ese mismo tiempo. A continuación, intentan determinar cuánto tardará el bote en viajar río abajo hasta dicha posición. Sin embargo, cuando el bote llegue a esa posición, el sombrero habrá continuado flotando corriente abajo. En consecuencia, la persona que está intentando resolver el problema trata de aplicar técnicas de cálculo numérico o se queda atrapado en un círculo vicioso intentando determinar dónde estará el sombrero cada vez que el bote llega al punto en que el sombrero se encontrara anteriormente.

Sin embargo, el problema es mucho más simple que esto. El truco está en resistirse a la tentación de comenzar inmediatamente a escribir fórmulas y a realizar cálculos. En su lugar, debemos dejar de lado estas habilidades y ajustar nuestra perspectiva. Todo el problema transcurre en el río. El hecho de que el agua se esté moviendo respecto a la orilla es irrelevante. Trate de pensar en el mismo problema como si tuviera lugar en una gran cinta transportadora en vez de en un río. Primero, resuelva el problema pensando en que la cinta transportadora está en reposo. Si colocamos el sombrero a nuestros pies mientras estamos en la cinta y luego nos alejamos de él andando durante 10 minutos, necesitaremos exactamente 10 minutos para volver a donde estaba el sombrero. Ahora, hagamos que la cinta transportadora se mueva. Esto quiere decir que la cinta se moverá con respecto al suelo y al edificio en el que se encuentre, pero como nosotros estamos sobre la cinta, esto no cambia nuestra relación ni con la cinta ni con el sombrero. Seguiremos necesitando 10 minutos para volver al punto en que estaba el sombrero.

Para concluir, digamos que el proceso de descubrimiento de algoritmos continúa siendo un arte de gran dificultad y que es preciso desarrollar a lo largo de un periodo de tiempo, más que ser un tema compuesto por metodologías bien definidas y que pueda enseñarse sin más. De hecho, tratar de entrenar a alguien que necesite dedicarse a la resolución de problemas, con el fin de que siga ciertas metodologías, termina constriñendo esas habilidades creativas que deberían, por el contrario, ser cultivadas.

Cuestiones y ejercicios

1. **a.** Encuentre un algoritmo para resolver el siguiente problema: dado un entero positivo n, halle la lista de enteros positivos cuyo producto sea el mayor de entre todas las listas cuya suma sea n. Por ejemplo, si n es 4, la lista deseada es 2, 2 porque 2×2 es mayor que $1 \times 1 \times 1 \times 1$, $2 \times 1 \times 1$ y 3×1. Si n es 5, la lista deseada es 2, 3.
   b. ¿Cuál será la lista deseada si $n = 2001$?
   c. Explique cómo ha conseguido abrirse camino en el problema.
2. **a.** Suponga que nos dan un tablero de ajedrez formado por 2^n filas y 2^n columnas, para un cierto n entero positivo, así como una caja de fichas con forma de L, cada una de las cuales puede cubrir exactamente tres cuadrados del tablero. Si quitamos del tablero un único cuadrado, ¿podemos cubrir el tablero restante con esas fichas, de modo que no se solapen ni se salgan por el borde del tablero?
   b. Explique cómo puede utilizarse su solución al apartado (a) para demostrar que $2^{2n} - 1$ es divisible por 3 para todos los enteros positivos n.
   c. ¿Cómo están relacionados los apartados (a) y (b) con las fases de Polya para la resolución de problemas?
3. Descifre el siguiente mensaje y explique luego cómo consiguió abrir camino en el problema. *Aopw ao hw ñaomqaopw ylññaypw.*
4. ¿Estaríamos siguiendo una metodología arriba-abajo si tratáramos de construir un puzzle simplemente disponiendo las piezas sobre una mesa e intentando juntarlas? ¿Cambiaría su respuesta si examináramos primero la imagen que viene en la caja del puzzle para ver cuál es el aspecto de la imagen completa?

5.4 Estructuras iterativas

Nuestro objetivo es estudiar ahora algunas de las estructuras repetitivas utilizadas para describir algoritmos. En esta sección vamos a analizar **estructuras iterativas** en las que un conjunto de sentencias se repite cíclicamente. En la siguiente sección presentaremos la técnica de la recursión. Como efecto colateral, también presentaremos algunos algoritmos muy populares: la búsqueda secuencial, la búsqueda binaria y la ordenación por inserción. Vamos a comenzar abordando el algoritmo de búsqueda secuencial.

Algoritmo de búsqueda secuencial

Considere el problema de buscar dentro de una lista para ver dónde aparece un determinado valor. Queremos desarrollar un algoritmo que determine si dicho valor se encuentra en la lista. Si el valor se encuentra en la lista, considerare-

mos que la búsqueda ha tenido éxito; en caso contrario, consideraremos que ha fallado. Vamos a presuponer que la lista está ordenada de acuerdo con algún tipo de regla para la ordenación de sus entradas. Por ejemplo, si la lista es una lista de nombres, supondremos que los nombres aparecen en orden alfabético, mientras que si la lista está compuesta por valores numéricos supondremos que sus entradas aparecen en orden creciente.

Para abrir camino en el problema, vamos a imaginar cómo buscaríamos en una lista de invitados compuesta por 20 nombres para tratar de encontrar a un invitado concreto. En ese caso, podríamos limitarnos a buscar en la lista desde el principio, comparando cada una de las entradas con el nombre que buscamos. Si encontramos el nombre deseado, la búsqueda termina y habrá tenido éxito; sin embargo, si alcanzamos el final de la lista sin encontrar el nombre buscado, la búsqueda termina habiendo fallado. De hecho, si al explorar la lista llegamos a un nombre que es posterior (alfabéticamente) al que estamos buscando sin haberlo encontrado, la búsqueda habrá terminado fallando (recuerde que la lista está ordenada alfabéticamente, por lo que llegar a un nombre posterior alfabéticamente al que estamos buscando indica que el que estamos buscando no se encuentra dentro de la lista). En resumen, nuestra idea general consiste en continuar buscando dentro de la lista mientras existan más nombres que comparar y mientras que el nombre buscado sea mayor, alfabéticamente, que el nombre con el que actualmente lo estamos comparando.

En nuestro pseudocódigo, este proceso puede representarse como

```
Seleccionar la primera entrada de la lista como
EntradaAComparar.
while (ValorObjetivo > EntradaAComparar y
       existan más entradas para comparar)
  do (Seleccionar la siguiente entrada de la lista
      como EntradaAComparar)
```

Al terminar esta estructura `while`, será cierta una de las dos condiciones siguientes: o bien hemos encontrado el valor objetivo o bien dicho valor objetivo no se encuentra en la lista. En cualquiera de los dos casos, podemos detectar si la búsqueda ha tenido éxito comparando la entrada a comparar con el valor objetivo. Si ambos valores son iguales, la búsqueda habrá tenido éxito. Por tanto, añadimos al final del pseudocódigo de la rutina la sentencia

```
if (ValorObjetivo = EntradaAComparar)
  then (Declarar que la búsqueda ha tenido éxito.)
  else (Declarar que la búsqueda ha fallado.)
```

Finalmente, observemos que la primera sentencia de nuestra rutina, que selecciona la primera entrada de la lista como la entrada a comparar, está basada en la suposición de que la lista en cuestión contiene al menos una entrada. Podríamos pensar que esta suposición es razonable, pero solo para asegurarnos podemos incluir nuestra rutina como la opción `else` de la sentencia

```
if (Lista vacía)
  then (Declarar que la búsqueda ha fallado.)
  else (. . .)
```

Esto nos da el procedimiento mostrado en la Figura 5.6. Observe que este se puede usar dentro de otros procedimientos mediante sentencias como

```
Aplicar el procedimiento Buscar a la lista de pasajeros
utilizando Juan González como valor objetivo.
```

para averiguar si Juan González es un pasajero y

```
Aplicar el procedimiento Buscar a la lista de ingredientes
utilizando nueces como valor objetivo.
```

para averiguar si las nueces aparecen en la lista de ingredientes.

En resumen, el algoritmo representado en la Figura 5.6 va tomando en consideración las entradas en el orden secuencial en el que aparecen dentro de la lista. Por esta razón, el algoritmo se denomina algoritmo de **búsqueda secuencial**. Debido a su simplicidad, se usa a menudo para listas cortas o cuando hay alguna otra razón que aconseje su uso. Sin embargo, en el caso de listas largas, las búsquedas secuenciales no son tan eficientes como otras técnicas (como pronto vamos a ver).

Control de bucles

El uso repetitivo de una sentencia o secuencia de sentencias es un concepto algorítmico importante. Un método para implementar dicha repetición es la estructura iterativa conocida como **bucle**, en la que un conjunto de sentencias denominado cuerpo del bucle, se ejecuta de forma repetida, bajo la dirección de algún tipo de proceso de control. Un ejemplo típico lo podemos encontrar en el algoritmo de búsqueda secuencial representado en la Figura 5.6. En él, utilizamos una sentencia `while` para controlar la repetición de la única instrucción `Seleccionar la siguiente entrada en la Lista como EntradaAComparar.` De hecho, la sentencia `while`

```
while (condición) do (cuerpo)
```

es un ejemplo del concepto de estructura de bucle, en el sentido de que su ejecución sigue el patrón típico

```
Comprobar la condición.
Ejecutar el cuerpo.
Comprobar la condición.
Ejecutar el cuerpo.
   .
   .
   .
Comprobar la condición.
```

hasta que la condición no se cumple.

Como regla general, el uso de una estructura iterativa permite tener un mayor grado de flexibilidad que el que obtendríamos si nos limitáramos a escribir explícitamente el cuerpo varias veces. Por ejemplo, para ejecutar la sentencia

```
Añadir una gota de ácido sulfúrico.
```

tres veces, escribiríamos:

```
Añadir una gota de ácido sulfúrico.
Añadir una gota de ácido sulfúrico.
Añadir una gota de ácido sulfúrico.
```

Figura 5.6 Algoritmo de búsqueda secuencial en pseudocódigo.

```
procedure Buscar (Lista, ValorObjetivo)
if (Lista vacía)
   then
     (Declarar que la búsqueda ha fallado.)
   else
     (Seleccionar la primera entrada de la Lista como EntradaAComparar;
      while (ValorObjetivo > EntradaAComparar y
               existan más entradas para comparar)
          do (Seleccionar la siguiente entrada de la Lista como EntradaAComparar.);
      if (ValorObjetivo = EntradaAComparar)
          then (Declarar que la búsqueda ha tenido éxito.)
          else (Declarar que la búsqueda ha fallado.)
     ) end if
```

Pero no podemos escribir una secuencia similar que sea equivalente a la secuencia de bucle

```
while (el nivel de pH sea mayor que 4) do
  (añadir una gota de ácido sulfúrico)
```

porque no sabemos de antemano cuántas gotas de ácido harán falta.

Examinemos ahora más de cerca los elementos de control del bucle. Podríamos sentirnos tentados a considerar que esta parte de la estructura de bucle tiene una importancia menor. Después de todo, normalmente es el cuerpo del bucle el que se encarga en realidad de llevar a cabo la tarea que tengamos entre manos (por ejemplo, añadir gotas de ácido), las actividades de control parecen simplemete las tareas administrativas necesarias para el proceso, porque hemos elegido ejecutar el cuerpo del bucle de forma repetitiva. Sin embargo, la experiencia demuestra que la parte de control de un bucle es la más proclive a la aparición de errores y merece por tanto que le dediquemos una atención especial.

El control de un bucle está compuesto por tres actividades: inicialización, comprobación y actualización (Figura 5.7), siendo imprescindible la presencia de esas tres actividades para poder controlar el bucle adecuadamente. La actividad de prueba se encarga de forzar la terminación del proceso de ejecución del bucle, examinando una condición que es la que indica que esa terminación debe producirse. Esta condición se conoce con el nombre de **condición de terminación** o **de finalización**. Es precisamente para esta comprobación para la

Figura 5.7 Componentes de la parte de control de un bucle.

Inicialización:	Establecer un estado inicial que será modificado hasta alcanzar la condición de terminación.
Comprobación:	Comparar el estado actual con la condición de terminación y finalizar la ejecución repetida del bucle en caso de que coincidan.
Actualización:	Cambiar el estado de tal manera que le haga progresar hasta la condición de terminación.

que proporcionamos una condición dentro de cada sentencia `while`. Sin embargo, en el caso de la instrucción `while`, la condición enunciada es la condición para que se ejecute el cuerpo del bucle; la condición de terminación será la negación de la condición que aparece en la estructura `while`. Por tanto, en la sentencia

```
while (el nivel de pH sea mayor que 4) do
 (añadir una gota de ácido sulfúrico)
```

la condición de terminación será "el nivel de pH ***no*** es mayor que 4", mientras que en la sentencia `while` de la Figura 5.6, la condición de terminación podría enunciarse como

(ValorObjetivo ≤ EntradaAComparar)
o (no hay más entradas para comparar)

Las otras dos actividades del control del bucle garantizan que la condición de terminación tenga lugar. El paso de inicialización establece una condición de partida y el paso de actualización hace que esta condición vaya progresando hasta alcanzar la condición de terminación. Por ejemplo, en la Figura 5.6, la inicialización tiene lugar en la sentencia que precede a la estructura `while`, donde se establece como entrada actual a comparar la primera entrada de la lista. El paso de actualización en este caso se lleva a cabo dentro del cuerpo del bucle, donde nuestro foco de interés (identificado por la entrada a comparar) se desplaza hacia el final de la lista. Así, habiendo ejecutado el paso de inicialización, la aplicación repetida del paso de actualización hace que se termine por alcanzar la condición de terminación (o bien llegamos a una entrada a comparar que sea mayor o igual que el valor objetivo o terminamos por alcanzar el final de la lista).

Hay que hacer hincapié en que los pasos de inicialización y actualización deben terminar conduciendo a la condición de terminación apropiada. Esta característica resulta crítica para un control adecuado del bucle y debemos, por tanto, comprobar exhaustivamente su presencia a la hora de diseñar una estructura iterativa. Si no hacemos esas comprobaciones, pueden aparecer errores incluso en los casos más simples. Un ejemplo típico es el que proporcionan las siguientes sentencias

```
Número ← 1;
while (Número ≠ 6) do
 (Número ← Número + 2)
```

La condición de terminación aquí es "`Número = 6`". Pero el valor de `Número` se inicializa a 1 y luego se incrementa en 2 unidades en el paso de actualización. Por tanto, a medida que vamos recorriendo el bucle, los valores asignados a `Número` serán 1, 3, 5, 7, 9, y así sucesivamente, pero nunca el valor 6. Debido a esto el bucle se ejecutará indefinidamente.

El orden en el que se ejecutan los distintos componentes del control del bucle puede tener consecuencias bastante sutiles. De hecho, existen dos estructuras iterativas comunes que difieren meramente en este aspecto. Un ejemplo de la primera sería la siguiente sentencia de pseudocódigo

```
while (condición) do (cuerpo)
```

cuya semántica se representa en la Figura 5.8 en forma de **diagrama de flujo** (estos diagramas emplean diversas formas geométricas para representar los pasos individuales y utilizan flechas para indicar el orden de los pasos. La distinción entre las diversas formas indica el tipo de acción implicada en el paso asociado. Un rombo indica una decisión, mientras que un rectángulo indica una sentencia o una secuencia de sentencias). Observe que la comprobación de si se ha alcanzado la condición de terminación en la estructura `while` tiene lugar *antes* de ejecutar el cuerpo del bucle.

Por el contrario, la estructura `repeat` de la Figura 5.9 exige que se ejecute el cuerpo del bucle antes de comprobar si se ha alcanzado la condición de terminación. En este caso, el cuerpo del bucle siempre se ejecuta *al menos una vez*, mientras que en la estructura `while`, el cuerpo nunca se ejecuta si se satisface la condición de terminación la primera vez que se comprueba.

Utilizaremos la forma sintáctica

```
repeat (cuerpo) until (condición)
```

en nuestro pseudocódigo para representar la estructura mostrada en la Figura 5.9. Así, la sentencia

```
repeat (sacar una moneda del bolsillo)
until (no haya más monedas en el bolsillo)
```

asume que tenemos al menos una moneda en el bolsillo al comenzar, mientras que

```
while (exista una moneda en el bolsillo) do
  (sacar una moneda del bolsillo)
```

no presupone que tengamos una moneda al empezar.

Siguiendo la terminología de nuestro pseudocódigo, normalmente nos referiremos a estas estructuras como bucle `while` o bucle `repeat`. En un contexto más genérico, puede que nos encontremos con que al bucle `while` se le denomina en ocasiones **bucle controlado a la entrada** (ya que la prueba de terminación se realiza antes de ejecutar el cuerpo del bucle) mientras que al bucle

Figura 5.8 Estructura del bucle while.

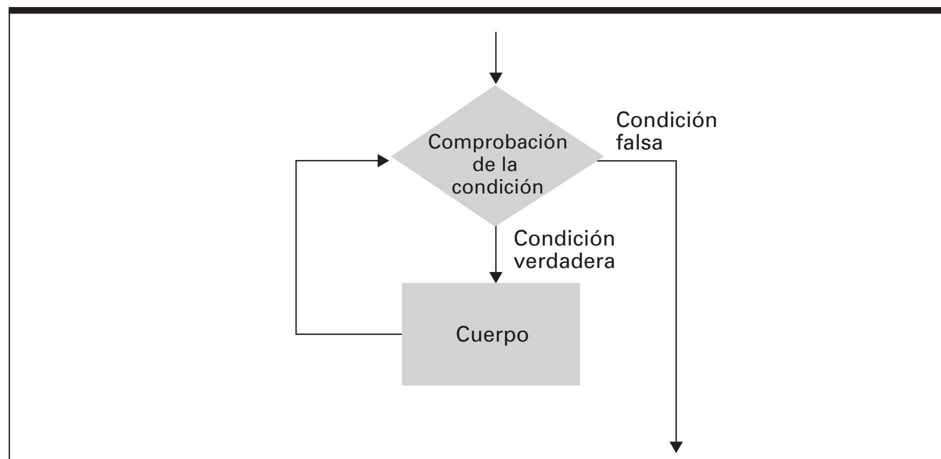
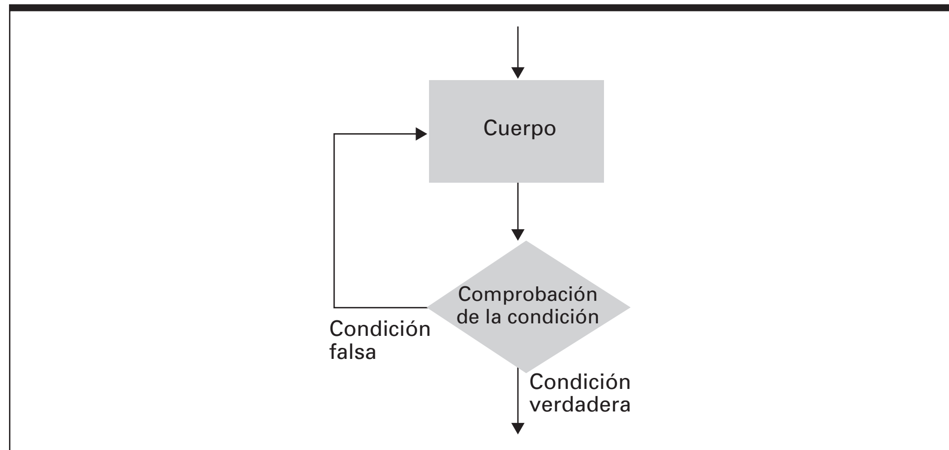


Figura 5.9 Estructura del bucle repeat.



`repeat` se le denomina **bucle controlado a la salida** (dado que la prueba de terminación se realiza después de haber ejecutado el cuerpo del bucle).

Algoritmo de ordenación por inserción

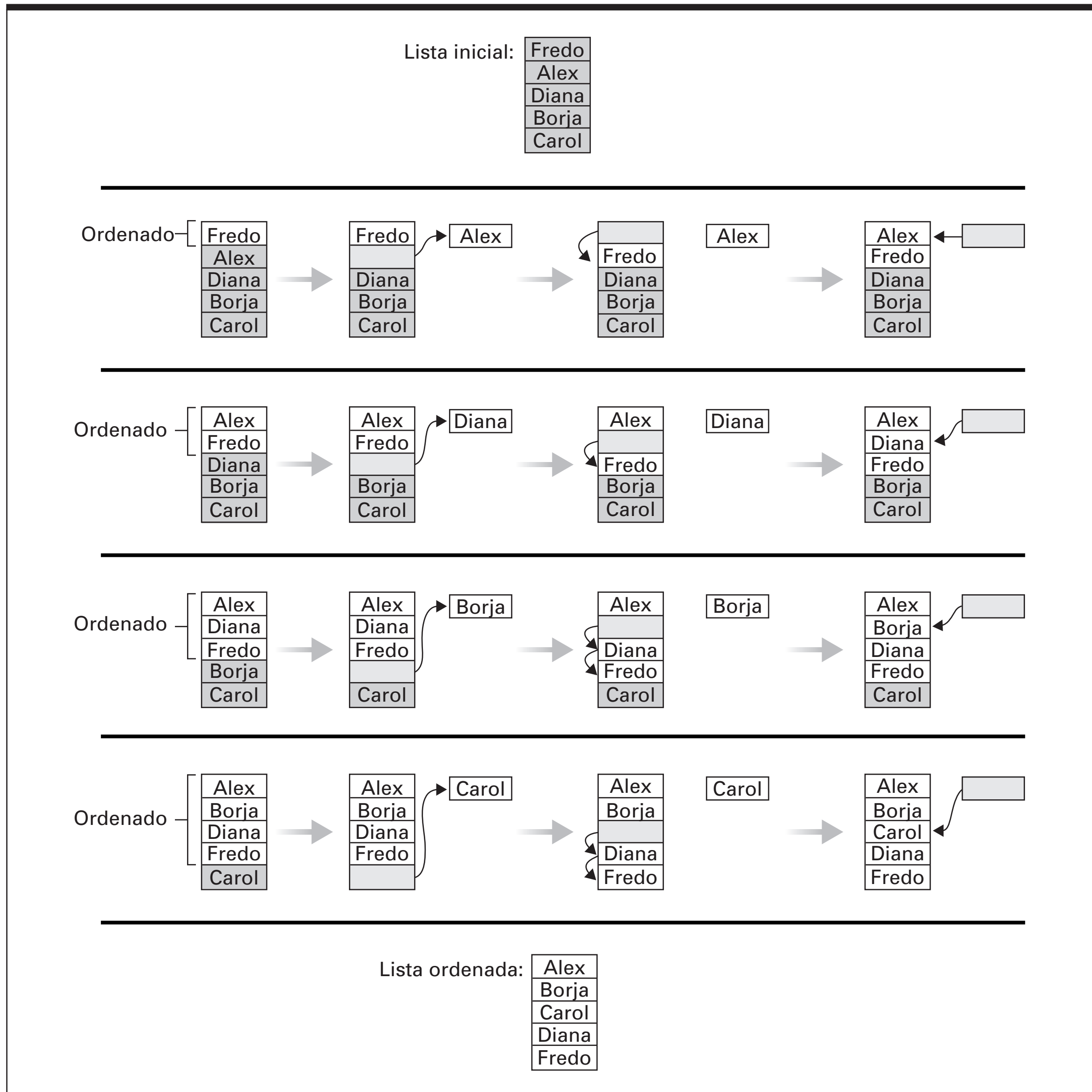
Como ejemplo adicional del uso de estructuras iterativas, vamos a considerar el problema de ordenar alfabéticamente una lista de nombres. Pero antes de continuar debemos identificar las restricciones con las que vamos a trabajar. Dicho de forma simple, nuestro objetivo es ordenar la lista "dentro de sí misma". En otras palabras, queremos ordenar la lista intercambiando sus entradas en lugar de ir odenando las entradas de la lista en una ubicación distinta. Nuestra situación es análoga al problema de ordenar una lista cuyas entradas estén anotadas en tarjetas de cartón independientes, distribuidas sobre una mesa atestada. Hemos despejado el suficiente espacio en la mesa como para poder disponer todas las tarjetas, pero no se nos permite empujar el resto del contenido de la mesa para poder disponer de más espacio. Esta restricción es muy típica en las aplicaciones informáticas, no porque el espacio de trabajo dentro de la máquina esté necesariamente atestado como nuestra mesa, sino simplemente porque deseamos emplear el espacio de almacenamiento disponible de la forma más eficiente.

Vamos a tratar de abrir brecha en la resolución del problema considerando cómo podríamos ordenar los nombres que tenemos sobre la mesa. Considere la siguiente lista de nombres:

Fredo
Alex
Diana
Borja
Carol

Una técnica para ordenar esta lista consiste en darse cuenta de que la sublista compuesta únicamente por el primero de los nombres, Fredo, está ordenada, mientras que la sublista compuesta por los dos primeros nombres, Fredo y Alex, no lo está. En consecuencia, podríamos tomar la tarjeta que contiene el

Figura 5.10 Ordenación alfabética de la lista de nombres Fredo, Alex, Diana, Borja y Carol .



nombre Alex, deslizar hacia abajo el nombre Fredo para que ocupe el espacio en el que se encontraba Alex y luego colocar el nombre Alex en el hueco que ha quedado en la parte superior de la lista, como se indica en la segunda fila de la Figura 5.10. En este punto nuestra lista sería

Alex
Fredo
Diana
Borja
Carol

Ahora, los dos primeros nombres forman una sublista ordenada, pero los tres primeros nombres no. Por tanto, debemos tomar el tercer nombre, Diana, deslizar el nombre Fredo hacia abajo hasta ocupar el hueco donde Diana se encontraba y luego insertar Diana en el hueco dejado por Fredo, como se ilustra en la tercera fila de la Figura 5.10. Ahora, las tres primeras entradas de la lista estarán ordenadas. Continuando de este modo, podemos obtener una lista en la que las cuatro primeras entradas estuvieran ordenadas, tomando el cuarto nombre, Borja, deslizando hacia abajo los nombres Fredo y Diana, y luego insertando Borja en el hueco (véase la cuarta fila de la Figura 5.10). Finalmente, podemos completar el proceso de ordenación tomando el nombre Carol, deslizando hacia abajo Fredo y Diana, e insertando después Carol en el hueco que ha quedado (véase la quinta fila de la Figura 5.10).

Habiendo analizado el proceso de ordenación de una lista concreta, nuestra tarea ahora consiste en generalizar este proceso con el fin de obtener un algoritmo para la ordenación de listas genéricas. Para ello, observemos que cada fila de la Figura 5.10 representa el mismo proceso general: seleccionar el primer nombre de la parte no ordenada de la lista, deslizar hacia abajo los nombres que sean posteriores alfabéticamente al nombre extraído y luego insertar otra vez en la lista el nombre extraído, en el lugar donde ha quedado un hueco. Si llamamos al nombre extraído con la denominación "entrada pivote", este proceso puede expresarse en nuestro pseudocódigo de la forma siguiente:

```
Mover la entrada pivote a una ubicación temporal,
dejando un hueco en la Lista;
while (exista un nombre por encima del hueco y
       dicho nombre sea mayor que el pivote) do
  (mover al hueco el nombre situado por encima del mismo,
   dejando un hueco encima de ese nombre)
Mover la entrada pivote al hueco de la Lista.
```

A continuación, observamos que este proceso debe ejecutarse de manera repetida. Para comenzar el proceso de ordenación, el pivote debe ser la segunda entrada de la lista y luego, antes de cada ejecución adicional, el pivote debe ser la entrada siguiente de la lista, hasta haber colocado adecuadamente la última entrada. Es decir, a medida que se repite la rutina que acabamos de describir, la posición inicial de la entrada pivote debe avanzar de la segunda entrada a la tercera, luego a la cuarta, etc., hasta que la rutina haya terminado de colocar la última entrada de la lista. Teniendo esto en cuenta, podemos controlar las repeticiones necesarias del bucle mediante las sentencias

```
N ← 2;
while (el valor de N no supere la longitud de Lista) do
   (Seleccionar la entrada N de la Lista como entrada
     pivote;
    .
    .
    .
    N ← N + 1)
```

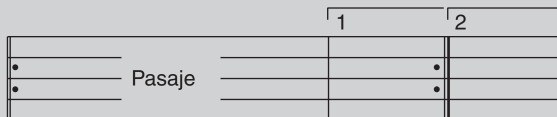
donde `N` representa la posición que hay que usar para la entrada pivote, la `longitud de Lista` hace referencia al número de entradas de la lista y los puntos indican dónde hay que ubicar la rutina anterior.

Estructuras iterativas en la música

Los músicos utilizaban y programaban estructuras iterativas muchos siglos antes de que lo hicieran los expertos en Ciencias de la computación. De hecho, la estructura de una canción (al estar compuesta por múltiples versos, cada uno de ellos seguido por el estribillo) puede expresarse mediante la sentencia while

```
while (quede algún verso) do
    (cantar el siguiente verso;
     cantar el estribillo)
```

Además, la notación



es meramente la forma que un compositor tiene de expresar la estructura

```
N ← 1;
while (N < 3) do
 (tocar el pasaje;
 tocar el N-ésimo final;
 N ← N + 1)
```

La Figura 5.11 muestra nuestro programa completo en pseudocódigo. En pocas palabras, el programa ordena una lista eliminando repetidamente una entrada e insertándola en la posición correcta. Es debido a este proceso repetido de inserción que el algoritmo subyacente se conoce como **ordenación por inserción**.

Observe que la estructura de la Figura 5.11 es la de un bucle dentro de otro bucle, estando el bucle exterior expresado por la primera sentencia `while` y el bucle interno por la segunda sentencia `while`. Cada ejecución del cuerpo del bucle exterior hace que el bucle interior se inicialice y se ejecute hasta que se alcance su condición de terminación. De esta manera, una única ejecución del

Figura 5.11 El algoritmo de ordenación por inserción en pseudocódigo.

```
procedure Ordenar (Lista)
N ← 2;
while (el valor de N no supera la longitud de Lista) do
    (Seleccionar la entrada N de Lista como entrada pivote;
     Mover la entrada pivote a una ubicación temporal dejando un hueco en Lista;
     while (haya un nombre por encima del hueco y dicho nombre es mayor que el pivote)
         do (mover al hueco el nombre situado por encima del mismo
             dejando un hueco por encima del nombre)
     Mover la entrada pivote al hueco de Lista;
     N ← N + 1
    )
```

cuerpo del bucle exterior provocará varias ejecuciones del cuerpo del bucle interior.

El componente de inicialización del bucle exterior consiste en establecer el valor inicial de N mediante la sentencia

```
N ← 2;
```

La actividad de actualización se gestiona incrementando el valor de N al final del cuerpo del bucle mediante la sentencia

```
N ← N + 1
```

La condición de terminación se alcanza cuando el valor de N supera la longitud de la lista.

El control del bucle interior se inicializa extrayendo la entrada pivote de la lista y creando así un hueco, el paso de modificación del bucle se lleva a cabo desplazando hacia abajo una serie de entradas para ocupar el hueco, con lo que ese hueco se va desplazando hacia arriba. La condición de terminación es que el hueco se encuentre justo debajo de un nombre que no sea superior al pivote o bien que el hueco alcance la parte superior de la lista.

Cuestiones y ejercicios

1. Modifique el procedimiento de búsqueda secuencial de la Figura 5.6 para que admita listas no ordenadas.
2. Convierta la rutina en pseudocódigo

```
Z ← 0;
X ← 1;
while (X < 6) do
  (Z ← Z + X;
   X ← X + 1)
```

   en una rutina equivalente que utilice un bucle `repeat`.
3. Algunos de los lenguajes de programación más populares de la actualidad utilizan la sintaxis

```
while (. . .) do (. . .)
```

   para representar un bucle controlado a la entrada y la sintaxis

```
do (. . .) while (. . .)
```

   para representar un bucle controlado a la salida. Aunque este diseño es elegante, ¿qué problemas podrían surgir debido a la similitud de ambas sintaxis?
4. Suponga que aplicáramos el procedimiento de ordenación por inserción presentado en la Figure 5.11 a la lista Genaro, Carmen, Alicia y Beatriz. Describa la organización de la lista al final de cada ejecución del cuerpo de la estructura `while` externa.
5. ¿Por qué no querríamos cambiar la frase "mayor que" en el bucle `while` de la Figura 5.11 por "mayor o igual que"?

6. Una variante del algoritmo de ordenación por inserción es la **ordenación por selección**. Esta variante comienza seleccionando la entrada más pequeña de la lista y moviéndola al principio de la misma. A continuación selecciona la entrada más pequeña de entre todas las entradas restantes de la lista y la mueve a la segunda posición de la lista. Seleccionando repetidamente la entrada más pequeña de la parte restante de la lista y moviendo hacia arriba esa entrada, la versión ordenada de la lista va aumentando a partir de la posición inicial de la misma, mientras que la parte posterior de la lista, compuesta por las entradas restantes no ordenadas, va reduciéndose. Utilice nuestro pseudocódigo para expresar un procedimiento similar al de la Figura 5.11 y que ordene una lista utilizando el algoritmo de ordenación por selección.

7. Otro algoritmo de ordenación muy conocido es el de **ordenación por burbuja**, la cual está basada en el proceso de comparar dos nombres adyacentes repetidamente e intercambiarlos si no se encuentran en el orden correcto. Supongamos que la lista en cuestión tiene n entradas. La ordenación por burbuja comenzaría comparando (y es posible que intercambiando) las entradas de las posiciones n y $n - 1$. A continuación, consideraría las entradas de las posiciones $n - 1$ y $n - 2$, y continuaría ascendiendo por la lista hasta que la primera y segunda entradas de la misma hubieran sido comparadas (y es posible que intercambiadas). Observe que esta pasada a través de la lista hará que la entrada más pequeña quede en la posición inicial. De forma similar, otra de esas pasadas hará que la segunda entrada más pequeña quede en la segunda posición de la lista. Así, haciendo un total de $n - 1$ pasadas a través de la lista, conseguiremos una lista completamente ordenada. (Si nos fijamos en el funcionamiento del algoritmo, parece que las entradas más pequeñas van ascendiendo como si fueran burbujas hacia la parte superior de la lista, de ahí su nombre.). Utilice nuestro pseudocódigo para expresar un procedimiento similar al de la Figura 5.11 para ordenar una lista utilizando el algoritmo de ordenación por burbuja.

5.5 Estructuras recursivas

Las estructuras recursivas proporcionan una alternativa al paradigma iterativo para la implementación de la repetición de actividades. Mientras que un bucle implica repetir un conjunto de sentencias de forma tal que el conjunto se completa y luego se repite, la recursión implica repetir el conjunto de sentencias como una subtarea de sí mismo. Como analogía, piense en el proceso de mantener conversaciones telefónicas mediante la funcionalidad de llamada en espera. En ese caso, se deja a un lado una conversación telefónica que aun no se ha completado mientras que se procesa otra llamada entrante. El resultado es que tienen lugar dos conversaciones. Sin embargo, no se llevan a cabo una detrás de la otra como sucedería en una estructura de bucle, sino que en su lugar una de las conversaciones se realiza dentro de la otra.

El algoritmo de búsqueda binaria

Con el fin de introducir la recursión, vamos a retomar el problema de buscar si existe una entrada concreta dentro de una lista ordenada, pero esta vez vamos a abrir brecha en el problema pensando en el procedimiento que normalmente seguimos a la hora de buscar en un diccionario. En este caso, no llevamos a cabo un procedimiento secuencial, entrada a entrada, ni siquiera página a página. En lugar de ello, comenzamos abriendo el diccionario por una página cualquiera, dentro del área donde pensamos que puede estar ubicada la entrada objetivo. Si tenemos suerte, encontraremos el valor objetivo en esa página; en caso contrario, debemos continuar buscando. Pero al llegar a ese punto habremos afinado nuestra búsqueda considerablemente.

Por supuesto, en el caso de la búsqueda dentro de un diccionario tenemos un conocimiento previo de dónde van a encontrarse las palabras. Si estamos buscando la palabra ***sonambulismo***, comenzaríamos abriendo el diccionario por una página situada cerca del final. Sin embargo, en el caso de las listas genéricas no tenemos esta ventaja inicial, así que vamos a decidir comenzar siempre nuestra búsqueda con la entrada "central" de la lista. Hemos escrito la palabra ***central*** entre comillas porque la lista puede tener un número par de entradas, en cuyo caso no existirá una entrada central en sentido estricto. En este caso, decidiremos arbitrariamente que la entrada "central" hace referencia a la primera entrada de la segunda mitad de la lista.

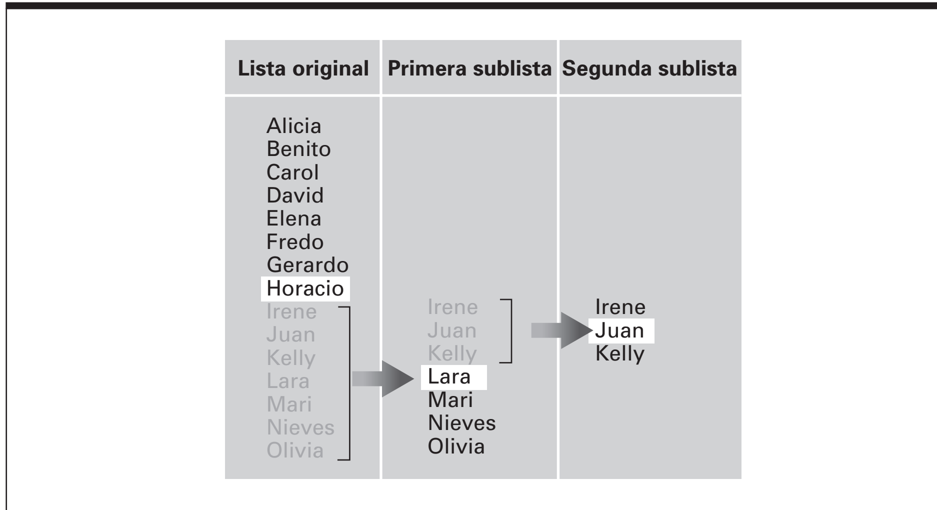
Si la entrada central de la lista es igual al valor buscado, podemos declarar que la búsqueda ha tenido éxito. En caso contrario, habremos restringido el proceso de búsqueda a la primera o a la segunda mitad de la lista, dependiendo de si el valor buscado es mayor o menor que la entrada que hemos analizado (recuerde que la lista está ordenada).

Para buscar en la parte restante de la lista, podríamos aplicar la búsqueda secuencial, pero en su lugar lo que haremos es aplicar a esa parte de la lista la misma técnica que hemos aplicado con la lista completa. Es decir, seleccionamos como siguiente entrada a considerar la entrada central de la parte restante de la lista. Como antes, si dicha entrada concuerda con el valor buscado, habremos terminado. En caso contrario, habremos restringido la búsqueda a una parte aún más pequeña de la lista.

Esta técnica de implementación del proceso de búsqueda se resume en la Figura 5.12, donde consideramos la tarea de buscar la entrada Juan en la lista de la parte izquierda de la figura. Consideraremos primero la entrada central, Horacio. Puesto que el valor que buscamos es posterior a esta entrada, continuáremos la búsqueda tomando en consideración la mitad inferior de la lista original. La entrada central de esta sublista es Lara. Puesto que nuestro valor objetivo es anterior a Lara, centraremos nuestra atención en la primera mitad de la sublista actual. Al comprobar el valor central de esa sublista secundaria nos encontramos con el valor buscado Juan y declaramos que la búsqueda ha tenido éxito. En resumen, la estrategia es dividir sucesivamente la lista en segmentos más pequeños, hasta encontrar el valor buscado o hasta restringir la búsqueda a un segmento vacío.

Es necesario recalcar este último punto. Si el valor buscado no se encuentra en la lista original, nuestra técnica de búsqueda dentro de la lista se desarrollará dividiendo la lista en segmentos más pequeños hasta que el segmento

Figura 5.12 Aplicación de nuestra estrategia a la búsqueda de la entrada Juan en una lista.



que estemos considerando esté vacío. Llegados a ese punto, el algoritmo tiene que ser capaz de reconocer que la búsqueda ha fallado.

La Figura 5.13 es un primer boceto de esta estrategia utilizando nuestro pseudocódigo. Comenzamos la búsqueda comprobando si la lista está vacía. En caso afirmativo, informaremos de que la búsqueda ha fallado. En caso contrario, consideraremos la entrada central de la lista. Si esta entrada no es igual al valor buscado, buscaremos en la primera o en la segunda mitad de la lista. Ambas posibilidades requieren realizar una búsqueda secundaria y, en ese sentido, sería muy conveniente poder realizar esas búsquedas invocando los servicios de una herramienta abstracta. En particular, nuestro enfoque consistirá en aplicar un procedimiento denominado `Buscar` para llevar a cabo estas búsquedas secundarias. Para completar el programa tendremos, por tanto, que proporcionar dicho procedimiento.

Figura 5.13 Un primer boceto de la técnica de búsqueda binaria.

```
if (Lista vacía)
 then
  (Informar de que la búsqueda ha fallado.)
 else
   [Seleccionar la entrada "central" de la Lista para que sea la EntradaAComparar;
    Ejecutar el bloque de instrucciones asociado
      con el caso apropiado.
        caso 1: ValorObjetivo = EntradaAComparar
               (Informar de que la búsqueda ha tenido éxito.)
        caso 2: ValorObjetivo < EntradaAComparar
               (Buscar ValorObjetivo en la parte de la Lista anterior a EntradaAComparar
                    e informar del resultado de dicha búsqueda.)
        caso 3: ValorObjetivo > EntradaAComparar
               (Buscar ValorObjetivo en la parte de la Lista posterior a EntradaAComparar
                    e informar del resultado de dicha búsqueda.)
   ] end if
```

Búsqueda y ordenación

Los algoritmos de búsqueda secuencial y binaria son simplemente dos de los múltiples algoritmos existentes para la implmentación del proceso de búsqueda. De la misma forma, la ordenación por inserción es uno de los múltiples algoritmos de ordenación que existen. Otros algoritmos clásicos de ordenación son, por ejemplo, la ordenación por combinación (de la que hablaremos en el Capítulo 12), la ordenación por selección (Cuestión/Ejercicio 6 de la Sección 5.4), la ordenación por burbuja (Cuestión/Ejercicio 7 de la Sección 5.4), la ordenación rápida (*quick sort*, que aplica una estrategia de tipo divide y vencerás al proceso de ordenación) y la ordenación por montículos (que utiliza una técnica inteligente para encontrar las entradas que deben moverse hacia arriba en la lista). Podrá encontrar descripciones de estos algoritmos en los libros que se enumeran en la sección Lecturas adicionales al final del capítulo.

Pero este procedimiento debe realizar la misma tarea que se expresa en el pseudocódigo que acabamos de escribir. Primero debe comprobar si la lista que se le ha proporcionado está vacía, y si no es así, debe analizar la entrada central de dicha lista. Por tanto, podemos suministrar el procedimiento necesario identificando simplemente la rutina actual como el procedimiento `Buscar` e insertando referencias a dicho procedimiento allí donde se requieran las búsquedas secundarias. El resultado se muestra en la Figura 5.14.

Observe que este procedimiento contiene referencias a sí mismo. Si estuviéramos ejecutando este procedimiento y nos encontráramos con la instrucción

```
Aplicar el procedimiento Buscar . . .
```

Figura 5.14 El algoritmo de búsqueda binaria en pseudocódigo.

```
procedure Buscar (Lista, ValorObjetivo)
if (Lista vacía)
   then
     (Informar de que la búsqueda ha fallado.)
   else
     [Seleccionar la entrada "central" de la Lista para que sea la EntradaAComparar;
      Ejecutar el bloque de instrucciones asociado
         con el caso apropiado.
            caso 1: ValorObjetivo= EntradaAComparar
                    (Informar de que la búsqueda ha tenido éxito.)
            caso 2: ValorObjetivo< EntradaAComparar
                    (Aplicar el procedimiento Buscar para ver si ValorObjetivo
                         está en la parte de la Lista anterior a EntradaAComparar,
                         e informar del resultado de dicha búsqueda.)
            caso 3: ValorObjetivo> EntradaAComparar
                    (Aplicar el procedimiento Buscar para ver si ValorObjetivo
                         está en la parte de la Lista posterior a EntradaAComparar,
                         e informar del resultado de dicha búsqueda.)
     ] end if
```

aplicaríamos al trozo de lista correspondiente el mismo procedimiento que ya estábamos aplicando a la lista original. Si dicha búsqueda tiene éxito volveríamos a nuestro procedimiento original y declararíamos que la búsqueda original ha tenido éxito; si esa búsqueda secundaria falla, declararíamos que la búsqueda original ha fallado.

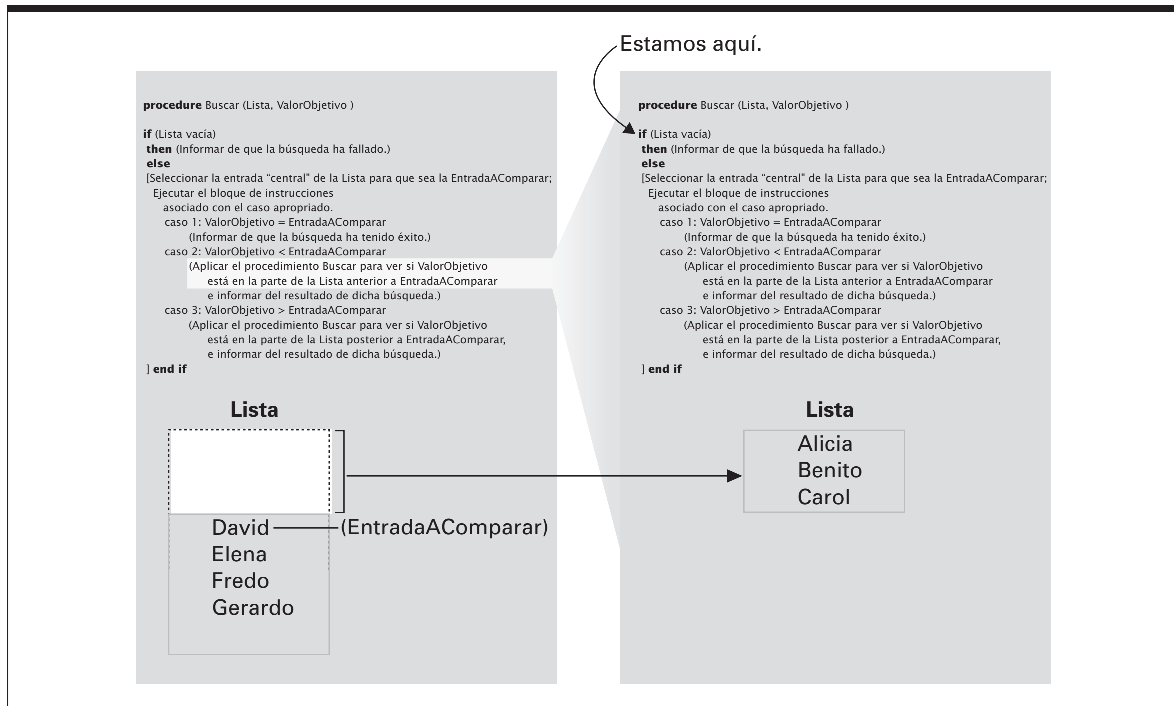
Para ver cómo lleva a cabo su tarea el procedimiento de la Figura 5.14, vamos a seguirlo mientras busca el valor Benito en la lista formada por los nombres Alicia, Benito, Carol, David, Elena, Fredo y Gerardo. Nuestra búsqueda comienza seleccionando David (la entrada central) como entrada a comparar. Puesto que el valor buscado (Benito) es anterior a la entrada seleccionada, tenemos que aplicar el procedimiento `Buscar` a la lista de entradas anterior a David, es decir, a la lista formada por los nombres Alicia, Benito y Carol. Al hacerlo, creamos una segunda copia del procedimiento `Buscar` y le asignamos esta tarea secundaria.

Ahora estaremos ejecutando dos copias del procedimiento de búsqueda, como se ilustra en la Figura 5.15. El progreso de la copia original se suspende en la sentencia

```
Aplicar el procedimiento Buscar para ver si
  ValorObjetivo está en la parte de la Lista
  anterior a EntradaAComparar
```

mientras aplicamos la segunda copia a la tarea de buscar en la lista formada por los nombres Alicia, Benito y Carol. Al completar esta búsqueda secundaria, descartamos la segunda copia del procedimiento, informamos del resultado del mismo a la copia original y continuamos con el procesamiento del procedi-

Figura 5.15



miento original. De esta forma, la segunda copia del procedimiento se ejecuta como una subordinada del original, realizando la tarea que el módulo original le ha solicitado y luego desapareciendo.

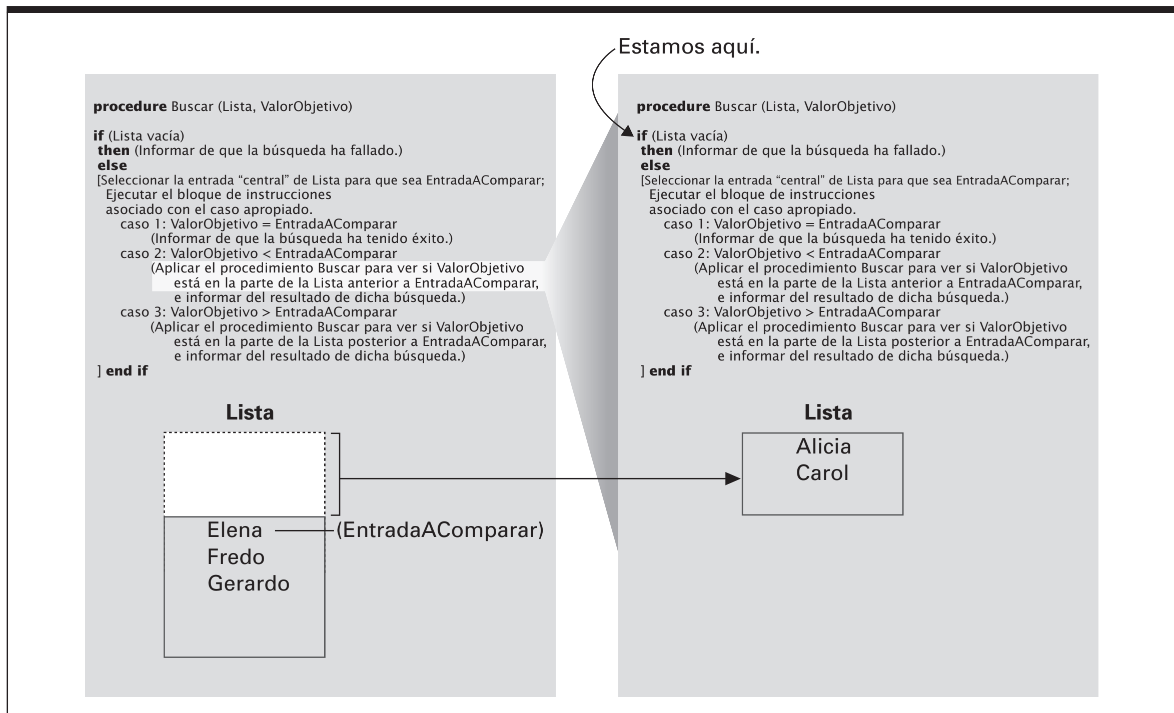
La búsqueda secundaria selecciona Benito como entrada a comparar porque esa es precisamente la entrada central de la lista formada por Alicia, Benito y Carol. Puesto que esta entrada coincide con el valor buscado, se declara que la búsqueda ha tenido éxito y el procedimiento termina. .

En este punto, habremos completado la búsqueda secundaria como nos había solicitado la copia original del procedimiento, así que podemos continuar con la ejecución de esa copia original. En ella se nos dice que el resultado de la búsqueda secundaria debe ser devuelto también como resultado de la búsqueda original. Por tanto, informamos de que la búsqueda original ha tenido éxito. Nuestro proceso ha determinado correctamente que Benito es un miembro de la lista formada por los nombres Alicia, Benito, Carol, David, Elena, Fredo y Gerardo.

Consideremos ahora lo que sucede si le pedimos al procedimiento de la Figura 5.14 que busque la entrada David en la lista formada por Alicia, Carol, Elena, Fredo y Gerardo. Esta vez, la copia original del procedimiento selecciona Elena como entrada a comprobar y concluye que el valor buscado debe encontrarse en la parte anterior de la lista. Por tanto, solicita a otra copia del procedimiento que busque entre la lista de entradas que aparece delante de Elena, es decir, en la lista de dos entradas compuesta por Alicia y Carol. En este momento, nuestra situación será la representada en la Figura 5.16.

La segunda copia del procedimiento selecciona Carol como entrada actual y concluye que el valor buscado debe estar en la parte posterior de su lista.

Figura 5.16

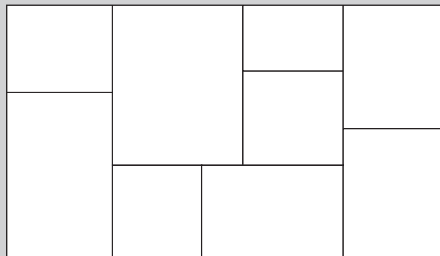


Entonces solicita a una tercera copia del procedimiento que busque entre la lista de nombres situados a continuación de Carol en la lista formada por Alicia y Carol. Esta sublista estará vacía, por lo que la tercera copia del procedimiento tiene como objetivo buscar el valor deseado, David, dentro de una lista vacía. Nuestra situación en este punto será la representada en la Figura 5.17. La copia original del procedimiento se encarga de buscar en la lista formada por los nombres Alicia, Carol, Elena, Fredo y Gerardo, siendo la entrada a comparar igual a Elena, la segunda copia se encarga de buscar en la lista formada por los nombres Alicia y Carol, siendo el valor a comparar igual a Carol y la tercera copia está a punto de comenzar a buscar en la lista vacía.

Por supuesto, la tercera copia del procedimiento declara enseguida que su búsqueda ha fallado y termina. La finalización de la tarea asignada a la tercera copia permite a la segunda copia continuar con su trabajo. Esa segunda copia observa que la búsqueda que ha solicitado no ha tenido éxito; en consecuencia, declara que su propia búsqueda ha fallado y termina. Este informe es lo que la copia original del procedimiento había estado esperando, por lo que ahora puede continuar. Puesto que la búsqueda solicitada ha fallado, declara que su propia búsqueda también ha fallado y termina. Nuestra rutina habrá concluido

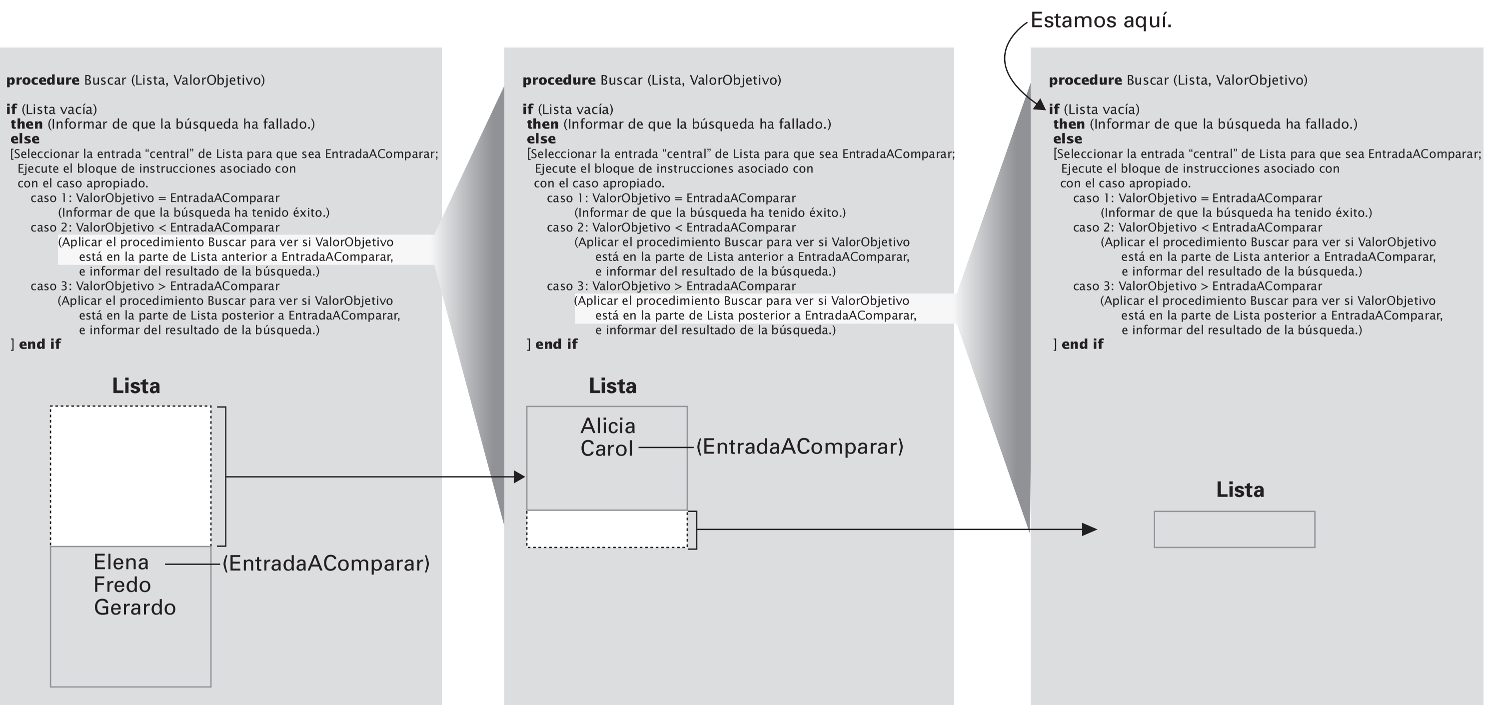
Estructuras recursivas en el arte

El siguiente procedimiento recursivo puede aplicarse a un lienzo rectangular, para generar dibujos del estilo del pintor holandés Piet Mondrian (1872–1944), que pintaba cuadros en los que el lienzo rectangular era dividido en rectángulos sucesivamente más pequeños. Pruebe a seguir el procedimiento usted mismo para generar dibujos similares al mostrado. Comience aplicando el procedimiento a un rectángulo que represente el lienzo en el que esté trabajando (si se está preguntando si el algoritmo representado por este procedimiento es un algoritmo que cumple con la definición de la Sección 5.1, sus sospechas están bien motivadas). Es, de hecho, un ejemplo de algoritmo no determinista, ya que hay momentos en los que a la persona o máquina que está siguiendo el procedimiento se le pide que tome decisiones "creativas". Quizá sea esta la razón por la que los resultados de Mondrian se consideran arte, mientras que los nuestros no.



procedure Mondrian (Rectángulo)
if (el tamaño de Rectángulo es demasiado para su gusto artístico)
  then (divida el Rectángulo en dos rectángulos más pequeños;
    aplicar el procedimiento Mondrian a uno de los rectángulos más pequeños;
    aplicar el procedimiento Mondrian al otro rectángulo más pequeño)

Figura 5.17


procedure Buscar (Lista, ValorObjetivo)
if (Lista vacía)
then (Informar de que la búsqueda ha fallado.)
else
[Seleccionar la entrada "central" de Lista para que sea EntradaAComparar;
Ejecute el bloque de instrucciones asociado con
con el caso apropiado.
caso 1: ValorObjetivo = EntradaAComparar
(Informar de que la búsqueda ha tenido éxito.)
caso 2: ValorObjetivo < EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista anterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
caso 3: ValorObjetivo > EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista posterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
] end if
Lista
Elena
Fredo
Gerardo
(EntradaAComparar)
procedure Buscar (Lista, ValorObjetivo)
if (Lista vacía)
then (Informar de que la búsqueda ha fallado.)
else
[Seleccionar la entrada "central" de Lista para que sea EntradaAComparar;
Ejecute el bloque de instrucciones asociado con
con el caso apropiado.
caso 1: ValorObjetivo = EntradaAComparar
(Informar de que la búsqueda ha tenido éxito.)
caso 2: ValorObjetivo < EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista anterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
caso 3: ValorObjetivo > EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista posterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
] end if
Lista
Alicia
Carol
(EntradaAComparar)
Estamos aquí.
procedure Buscar (Lista, ValorObjetivo)
if (Lista vacía)
then (Informar de que la búsqueda ha fallado.)
else
[Seleccionar la entrada "central" de Lista para que sea EntradaAComparar;
Ejecute el bloque de instrucciones asociado con
con el caso apropiado.
caso 1: ValorObjetivo = EntradaAComparar
(Informar de que la búsqueda ha tenido éxito.)
caso 2: ValorObjetivo < EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista anterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
caso 3: ValorObjetivo > EntradaAComparar
(Aplicar el procedimiento Buscar para ver si ValorObjetivo
está en la parte de Lista posterior a EntradaAComparar,
e informar del resultado de la búsqueda.)
] end if
Lista

correctamente que David no está en la lista de nombres formada por Alicia, Carol, Elena, Fredo y Gerardo.

En resumen, si analizamos de nuevo los procesos anteriores, podemos ver que el algoritmo representado en la Figura 5.14 consiste en dividir repetidamente la lista en cuestión en dos fragmentos más pequeños de forma tal que la búsqueda restante pueda restringirse a uno solo de esos fragmentos. Esta técnica de división en dos es la razón por la que el algoritmo se conoce con el nombre de **búsqueda binaria**.

Control recursivo

El algoritmo de búsqueda binaria es similar al de búsqueda secuencial, en el sentido de que cada uno de estos algoritmos solicita la ejecución repetida de un proceso. Sin embargo, la implementación de estas repeticiones es significativamente diferente. Mientras que la búsqueda secuencial implica una repetición cíclica, la búsqueda binaria ejecuta cada paso de la repetición como una subtarea del paso anterior. Esta técnica se conoce con el nombre de **recursión**.

Como hemos visto, la ilusión creada por la ejecución de un procedimiento recursivo es que existen múltiples copias del procedimiento, cada una de las cuales se denomina activación del procedimiento. Estas activaciones se crean dinámicamente de forma “telescópica” y terminan por desaparecer a medida que el algoritmo progresa. De las activaciones existentes en cualquier momento determinado, solo una está progresando activamente. Las otras se encuentran en la práctica en una especie de limbo, esperando cada una de ellas a que otra activación termine antes de poder continuar.

Tratándose de procesos repetitivos, los sistemas recursivos dependen tanto de la existencia de estructuras de control apropiadas como los bucles. Al igual que sucede en el control de un bucle, los sistemas recursivos dependen de que se compruebe adecuadamente la condición de finalización y de que se adopte un diseño que garantice que dicha condición será alcanzada en algún momento. De hecho, un adecuado control recursivo necesita de los tres mismos ingredientes (inicialización, actualización y comprobación de terminación), que se requieren en el control de un bucle.

En general, un procedimiento recursivo se diseña para comprobar si se ha alcanzado la condición de terminación (que a menudo se denomina **caso base** o **caso degenerado**) antes de solicitar ulteriores activaciones. Si no se cumple la condición de finalización, la rutina crea otra activación del procedimiento y le asigna la tarea de resolver un subproblema que se encuentre más próximo a la condición de terminación que el problema asignado a la activación actual. Sin embargo, si se cumple la condición de terminación, se toma un camino que hace que la activación actual termine sin crear activaciones adicionales.

Veamos cómo se implementan las fases de inicialización y actualización para el control de las repeticiones en nuestro procedimiento de búsqueda binaria de la Figura 5.14. En este caso, la creación de activaciones adicionales se termina una vez que se encuentra el valor buscado o si la tarea se reduce a buscar en una lista vacía. El proceso se inicializa de manera implícita, cuando se le proporciona una lista inicial y un valor objetivo. A partir de esta configuración

inicial, el procedimiento modifica su tarea asignada, transformándola en la de buscar en una lista más pequeña. Puesto que la lista original tiene una longitud finita y en cada paso de modificación se reduce la longitud de la lista en cuestión, podemos estar seguros de que el valor objetivo terminará encontrándose o de que la tarea se reducirá a buscar en una lista vacía. Podemos concluir por tanto que existe la garantía de que el proceso repetitivo termine.

Finalmente, puesto que tanto las estructuras de control iterativo como recursivo son formas de provocar la repetición de un conjunto de sentencias, podríamos preguntarnos si ambas tienen una potencia equivalente: es decir, si diseñamos un algoritmo utilizando una estructura iterativa, ¿podría diseñarse otro algoritmo que utilice solo técnicas recursivas y que resuelve el mismo problema, y viceversa? Esas preguntas son importantes en las Ciencias de la computación, porque su respuesta nos dice, en último término, qué características deben ofrecerse en un lenguaje de programación para poder obtener el sistema de programación más potente posible. Volveremos sobre estas ideas en el Capítulo 12, en el que consideraremos algunos de los aspectos más teóricos de las Ciencias de la computación y de su base matemática. Con esta información, seremos entonces capaces de demostrar la equivalencia de las estructuras iterativas y recursivas en el Apéndice E.

Cuestiones y ejercicios

1. ¿Qué nombres serán seleccionados para comparar por el algoritmo de búsqueda binaria (Figura 5.14) cuando estemos buscando el nombre Jose en la lista de nombres Alicia, Beatriz, Carol, Diana, Elena, Fredo, Gerardo, Hector, Irene, Jose, Katy, Lara, Mari, Nieves y Olivia?
2. ¿Cuál es el número máximo de entradas que habrá que comprobar al aplicar la búsqueda binaria a una lista de 200 entradas? ¿Y qué sucede con una lista de 100.000 entradas?
3. ¿Qué secuencia de números imprimiría el siguiente procedimiento recursivo si comenzáramos asignando a N el valor 1?

```
procedure Ejercicio (N)
imprimir el valor de N;
if (N < 3) then (aplicar el procedimiento Ejercicio al
                 valor N + 1);
imprimir el valor de N.
```

4. ¿Cuál es la condición de finalización en el procedimiento recursivo de la Cuestión/Ejercicio 3?

5.6 Eficiencia y corrección

En esta sección presentamos dos temas que constituyen áreas de investigación importantes dentro del campo de las Ciencias de la computación. El primero de ellos es la eficiencia de los algoritmos y el segundo es el de su corrección.

Eficiencia de un algoritmo

Aunque las máquinas actuales son capaces de ejecutar millones de instrucciones por segundo, la eficiencia continúa siendo una de las preocupaciones principales en el diseño de algoritmos. A menudo la elección entre un algoritmo eficiente y otro ineficiente puede marcar la diferencia entre una solución práctica a un problema y otra completamente inútil.

Consideremos el problema de la persona encargada de realizar las matrículas en una universidad, quien tiene la tarea de obtener y actualizar los registros de los estudiantes. Aunque la universidad tiene aproximadamente 10.000 estudiantes en cualquier semestre, su archivo de "estudiantes actuales" contiene los registros de más de 30.000 alumnos que se consideran actuales en el sentido de que se han matriculado para asistir al menos a un curso en los últimos años, pero aun no han completado su carrera. Por ahora, vamos a suponer que estos registros están almacenados en la computadora del departamento de administración en forma de una lista ordenada según el número de identificación de los estudiantes. Para encontrar el registro de cualquier estudiante, el encargado buscaría en esta lista un determinado número de identificación.

Hemos presentado dos algoritmos para buscar en una lista de ese tipo: la búsqueda secuencial y la búsqueda binaria. Nuestra pregunta ahora es si la elección entre estos dos algoritmos tiene alguna importancia para la persona encargada del registro. Vamos a considerar en primer lugar la búsqueda secuencial.

Dado un número de identificación de un estudiante, el algoritmo de búsqueda secuencial comienza por el principio de la lista y compara las entradas que va extrayendo con el número de identificación deseado. Al no saber nada acerca del origen del valor seleccionado como objetivo, no podemos deducir qué parte de la lista habrá que recorrer en esta búsqueda. Lo que sí podemos decir, no obstante, es que después de muchas búsquedas cabe esperar que la profundidad media de esas búsquedas sea igual a la mitad del tamaño de la lista. Algunas de las búsquedas serán más cortas, pero otras serán más largas. Por tanto, podemos estimar que a lo largo de un cierto periodo de tiempo, la búsqueda secuencial analizará aproximadamente 15.000 registros por búsqueda. Si el obtener y comprobar cada registro para ver su número de identificación requiere 10 milisegundos (10 milésimas de segundo), dicha búsqueda requeriría una media de 150 segundos o 2,5 minutos (un tiempo insoportablemente largo para la persona encargada del registro que tiene que esperar a que aparezca el registro del estudiante en la pantalla de su computadora). Incluso si el tiempo requerido para extraer y comprobar cada registro se redujera a solo 1 milisegundo, la búsqueda seguiría necesitando una media de 15 segundos, que continúa representando un largo tiempo de espera.

Por el contrario, la búsqueda binaria funciona comparando el valor objetivo con la entrada central de la lista. Si no coinciden ambos valores, entonces al menos la búsqueda se habrá reducido a solo la mitad de la lista original. Así, después de comprobar la entrada central de la lista de 30.000 registros de estudiantes, la búsqueda binaria tendrá que tener en cuenta como máximo 15.000 registros. Después de la segunda comparación, quedarán como máximo 7.500 y después de la tercera consulta, la lista en cuestión se habrá reducido a no más de 3.750 entradas. Continuando de esta forma, podemos ver que se podrá encontrar el registro que andamos buscando después de extraer como máximo

15 entradas de la lista de 30.000 registros. Por tanto, si cada una de estas extracciones puede realizarse en 10 milisegundos, el proceso de búsqueda de un registro concreto requiere solo 0,15 segundos, lo que quiere decir que el acceso al registro de un estudiante concreto parecerá ser instantáneo desde el punto de vista de la persona encargada de los registros. Podemos concluir entonces que la elección entre el algoritmo de búsqueda secuencial y el de búsqueda binaria tendría un impacto significativo en esta aplicación.

Este ejemplo indica la importancia del área de las Ciencias de la computación conocida como análisis de algoritmos y que abarca el estudio de los recursos, como por ejemplo el tiempo o el espacio de almacenamiento, que los algoritmos requieren. Una de las principales aplicaciones de este tipo de estudios es la evaluación de las ventajas relativas de los distintos algoritmos existentes para resolver un problema concreto.

El análisis de algoritmos implica a menudo analizar los escenarios de caso mejor, caso peor y caso promedio. En nuestro ejemplo hemos realizado un análisis del caso promedio del algoritmo de búsqueda secuencial y un análisis de caso peor del algoritmo de búsqueda binaria, con el fin de estimar el tiempo requerido para buscar a través de una lista con 30.000 entradas. En general, dicho análisis se hace en un contexto más genérico; es decir, a la hora de considerar los distintos algoritmos existentes para realizar búsquedas en listas, no nos centramos en una lista de longitud concreta, sino que en su lugar tratamos de identificar una fórmula que nos indique el rendimiento del algoritmo para listas de longitud arbitraria. No es difícil generalizar nuestros razonamientos anteriores al caso de listas de cualquier longitud. En particular, cuando aplicamos esos razonamientos a una lista con n entradas, el algoritmo de búsqueda secuencial tendrá que consultar una media de $n/2$ entradas, mientras que el algoritmo de búsqueda binaria tendrá que consultar como máximo $\lg n$ entradas en el escenario del caso peor. ($\lg n$ representa el logaritmo en base dos de n.)

Analicemos el algoritmo de búsqueda por inserción (ilustrado en la Figura 5.11) de una forma similar. Recuerde que este algoritmo implica seleccionar una entrada de la lista, denominada entrada pivote, comparar dicha entrada con las que la preceden hasta encontrar el lugar adecuado para el pivote, y luego insertar la entrada pivote en ese lugar. Puesto que la actividad de comparar dos entradas es la predominante en este algoritmo, nuestra técnica consistirá en contar el número de esas comparaciones que se realizan al ordenar una lista cuya longitud es n.

El algoritmo comienza seleccionando como pivote la segunda entrada de la lista. Después continúa seleccionando como pivote las entradas sucesivas hasta alcanzar el final de la lista. En el mejor de los casos posibles, cada pivote ya se encontrará en su lugar adecuado, con lo que solo hará falta compararlo con una única entrada antes de descubrir que está bien colocado. En consecuencia, en el mejor de los casos, el aplicar la ordenación por inserción a una lista con n entradas requiere $n - 1$ comparaciones. (La segunda entrada se compara con una entrada, la tercera entrada se compara también con una entrada, y así sucesivamente.)

Por el contrario, el escenario de caso peor es que cada pivote tenga que compararse con todas las entradas precedentes para poder encontrar la ubicación correcta. Esto sucede si la lista original está en orden inverso. En este

caso, el primer pivote (la segunda entrada de la lista) se compara con una entrada, el segundo pivote (la tercera entrada de la lista) se compara con dos entradas, y así sucesivamente (Figura 5.18). Por tanto, el número total de comparaciones al ordenar una lista de n entradas será $1 + 2 + 3 + \ldots + (n - 1)$, que es equivalente a $(1/2)(n^2 - n)$. En particular, si la lista contuviera 10 entradas, el escenario de caso peor para el algoritmo de inserción por ordenación requeriría 45 comparaciones.

En el caso promedio de la ordenación por inserción, cabría esperar que cada pivote tuviera que compararse con la mitad de las entradas que lo preceden. Esto hace que haya que realizar la mitad de las comparaciones que en el caso peor, es decir un total de $(1/4)(n^2 - n)$ comparaciones para ordenar una lista de n entradas. Por ejemplo, si utilizamos la ordenación por inserción para ordenar diversas listas de longitud 10, podemos esperar que el número medio de comparaciones por cada ordenación sea de 22,5.

La importancia de estos resultados es que el número de comparaciones realizadas durante la ejecución del algoritmo de ordenación por inserción nos da una aproximación de la cantidad de tiempo requerida para ejecutar el algoritmo. Utilizando esta aproximación, la Figura 5.19 muestra una gráfica que indica cómo se incrementa el tiempo requerido para ejecutar el algoritmo de ordenación por inserción a medida que aumenta la longitud de la lista. Esta gráfica está basada en el análisis del caso peor del algoritmo, en el que habíamos concluido que el ordenar una lista de longitud n requeriría como máximo $(1/2)(n^2 - n)$ comparaciones entre las entradas de la lista. En la gráfica, hemos marcado diversas longitudes de lista y hemos indicado el tiempo requerido en cada caso. Observe que, a medida que la longitud de las listas se incrementa en pasos uniformes, el tiempo requerido para ordenar la lista se incrementa en una cantidad cada vez mayor. Por tanto, el algoritmo es menos eficiente a medida que va aumentando el tamaño de la lista.

Apliquemos un análisis similar al algoritmo de búsqueda binaria. Recuerde que hemos concluido que el buscar en una lista con n entradas utilizando este algoritmo requiere comparar como máximo $\lg n$ entradas, lo que de nuevo nos da una aproximación de la cantidad de tiempo requerida para ejecutar el algoritmo para diversos tamaños de lista. La Figura 5.20 muestra una gráfica basada en este análisis, en la que de nuevo hemos marcado diversas longitudes de lista con incrementos uniformes y hemos identificado el tiempo que requiere el algoritmo en cada caso. Observe que el tiempo requerido por el algoritmo se

Figura 5.18 Aplicación de la ordenación por inserción en el caso peor.

Lista inicial	Comparaciones realizadas para cada pivote: 1er pivote	2º pivote	3er pivote	4º pivote	Lista ordenada
Elena	1 Elena	3 David	6 Carol	10 Barbara	Alfredo
David	David	2 Elena	5 David	9 Carol	Barbara
Carol	Carol	Carol	4 Elena	8 David	Carol
Barbara	Barbara	Barbara	Barbara	7 Elena	David
Alfredo	Alfredo	Alfredo	Alfredo	Alfredo	Elena

Figura 5.19 Gráfica del análisis de caso peor para el algortimo de ordenación por inserción.

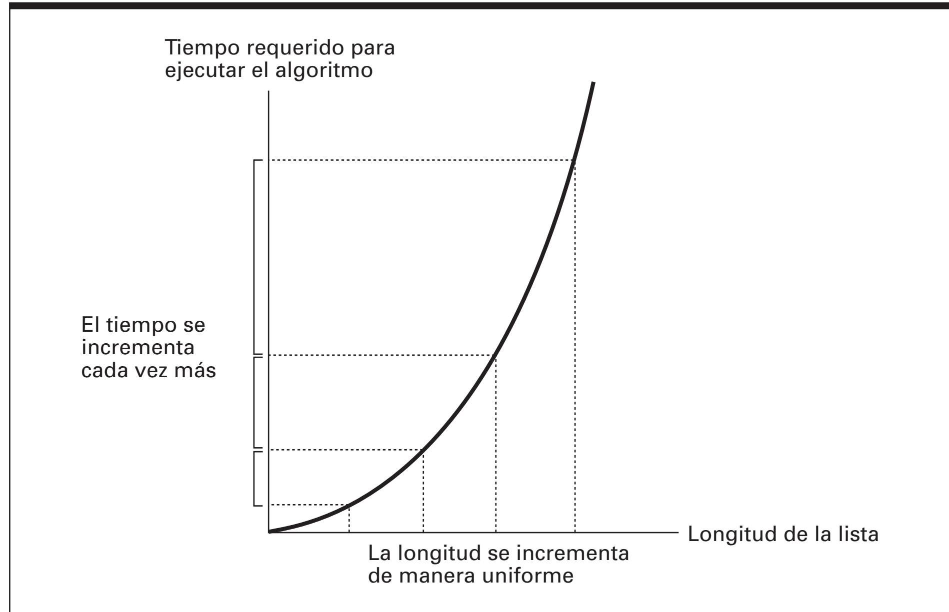
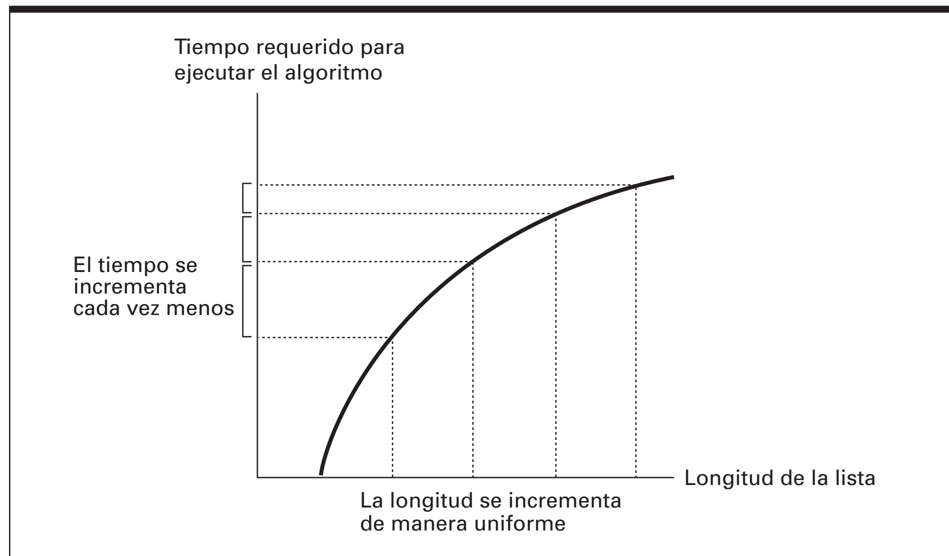


Figura 5.20 Gráfica del análisis de caso peor para el algoritmo de búsqueda binaria.



incrementa cada vez menos. Es decir, el algoritmo de búsqueda binaria es más eficiente a medida que se va incrementando el tamaño de la lista.

El factor distintivo entre las Figuras 5.19 y 5.20 es la forma general de las gráficas que aparecen. Esta forma general revela lo bien que cabe esperar que se comporte un algoritmo a medida que las entradas van teniendo un tamaño cada vez mayor. Además, la forma general de una gráfica está determinada por el tipo de expresión a la que se está representando, más que por los deta-

lles específicos de la expresión (todas las fórmulas de tipo lineal generan una línea recta; todas las expresiones de tipo cuadrático generan una curva parabólica; todas las fórmulas logarítmicas producen la gráfica logarítmica mostrada en la Figura 5.20). Es habitual identificar una forma utilizando la más simple de las fórmulas que genera dicha forma. En particular, identificamos la forma parabólica mediante la fórmula n^2 y la forma logarítmica mediante la expresión ***lg n***.

Dado que la forma de la gráfica obtenida al comparar el tiempo requerido por un algoritmo para realizar su tarea con el tamaño de los datos de entrada refleja las características de eficiencia del algoritmo, es común clasificar los algoritmos de acuerdo con las formas de estas gráficas, basándose normalmente en el análisis de caso peor del algoritmo. La notación utilizada para identificar estas clases se denomina, en ocasiones, **notación zeta-mayúscula**. Todos los algoritmos cuyas gráficas tienen la forma de una parábola, como el de ordenación por inserción, se adjudican a la clase representada por $\Theta(n^2)$ (lo que se lee "zeta mayúscula de n al cuadrado"); todos los algoritmos cuyas gráficas tienen la forma de una expresión logarítmica, como la búsqueda binaria, caen dentro de la clase representada por $\Theta(lg\ n)$ (que se lee "zeta mayúscula de ***log n***"). Conociendo la clase a la que pertenece un algoritmo concreto podemos predecir su rendimiento y compararlo con otros algoritmos que resuelven el mismo problema. Dos algoritmos de tipo $\Theta(n^2)$ exhibirán cambios similares en sus necesidades de tiempo a medida que se incremente el tamaño de las entradas. Además, las necesidades de tiempo de un algoritmo de tipo $\Theta(lg\ n)$ no aumentarán tan rápidamente como las de un algoritmo de tipo $\Theta(n^2)$.

Verificación del software

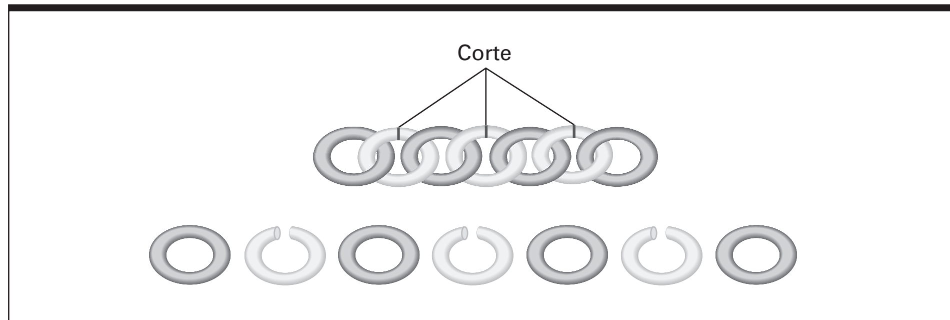
Recuerde que la cuarta fase del análisis de Polya para la resolución de problemas (Sección 5.3) consiste en evaluar la solución para verificar su precisión y para ver su potencial como herramienta para la resolución de otros problemas. La importancia de la primera parte de esta fase queda ilustrada mediante el siguiente ejemplo:

> Un viajante con una cadena de oro compuesta por siete eslabones debe permanecer en un hotel aislado durante siete noches. Para pagar la habitación, cada noche tiene que entregar un eslabón de la cadena. ¿Cuál es el menor número de eslabones que habrá que cortar para que el viajante pueda pagar el hotel entregando un eslabón de la cadena cada mañana, sin pagar su estancia por adelantado?

Para resolver este problema, nos damos cuenta en primer lugar de que no hace falta cortar todos los eslabones de la cadena. Si cortamos solo el segundo eslabón, podremos separar tanto el primer eslabón como el segundo de los otros cinco. Teniendo esto en cuenta, llegamos a la solución de que solo es necesario cortar el segundo, el cuarto y el sexto eslabones de la cadena, lo cual es un proceso que permite liberar todos los eslabones mientras que solo se cortan tres de ellos (Figura 5.21). Además, cualquier intento de solución con menos cortes dejará siempre dos eslabones unidos, por lo que podemos concluir que la respuesta correcta a nuestro problema es tres.

Sin embargo, al reconsiderar el problema, podemos hacer la observación de que cuando solo se corta el tercer eslabón de la cadena, obtenemos tres frag-

Figura 5.21 Separación de la cadena utilizando solo tres cortes.



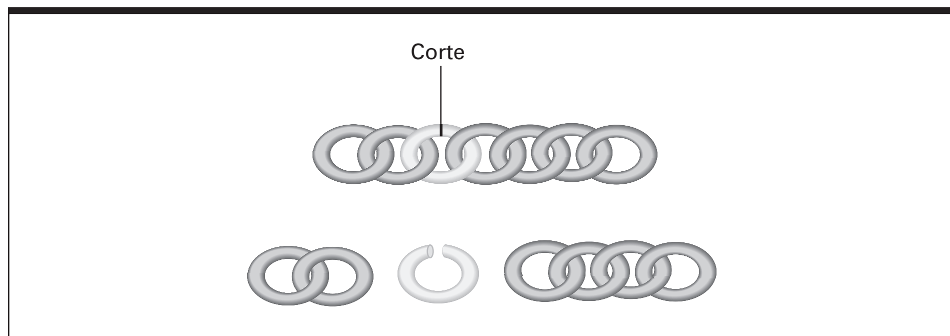
mentos de cadena cuyas longitudes respectivas son uno, dos y cuatro (Figura 5.22). Con estos fragmentos podemos proceder de la forma siguiente:

> Primera mañana: Dar al hotel el eslabón aislado.
>
> Segunda mañana: Pedir que nos devuelvan el eslabón aislado y dar al hotel el trozo cadena con dos eslabones.
>
> Tercera mañana: Dar al hotel el eslabón aislado.
>
> Cuarta mañana: Pedir que nos devuelvan los tres eslabones que tiene y darle el trozo de cadena de cuatro eslabones.
>
> Quinta mañana: Dar al hotel el eslabón aislado.
>
> Sexta mañana: Pedir que nos devuelvan el eslabón aislado y darle el trozo de cadena con dos eslabones.
>
> Séptima mañana: Dar al hotel el eslabón aislado.

En consecuencia, nuestra primera respuesta, que pensábamos que era correcta, no lo es. ¿Cómo podemos entonces estar seguros de que nuestra nueva solución es correcta? Podríamos argumentar de la forma siguiente: puesto que hay que darle al hotel un único eslabón en la primera mañana, es necesario cortar al menos un eslabón de la cadena, y puesto que nuestra nueva solución requiere un único corte, debe ser óptima.

Traducido al entorno de programación, este ejemplo resalta la distinción entre un programa que se cree que es correcto y un programa que verdaderamente lo es. Las dos cosas no necesariamente coinciden. La comunidad de procesamiento de datos cuenta con numerosas historias de horror relativas a software que se "sabía" que era correcto y que sin embargo falló en un mo-

Figura 5.22 Resolución del problema mediante un único corte.



mento crítico debido a una situación imprevista. Por tanto, la verificación del software es una tarea de gran importancia, y la búsqueda de técnicas eficientes de verificación constituye un campo de investigación muy activo dentro de las Ciencias de la computación.

Una de las principales líneas de investigación en esta área trata de aplicar las técnicas de la lógica formal para demostrar la corrección de un programa. Es decir, el objetivo es aplicar la lógica formal para demostrar que el algoritmo representado por un programa hace lo que se pretende que haga. La tesis subyacente es que, al reducir el proceso de verificación a un procedimiento formal, estamos protegidos frente a las conclusiones incorrectas que pudieran estar asociadas con los argumentos intuitivos, como era el caso del problema de la cadena de oro. Consideremos con más detalle este enfoque para la verificación de programas.

Al igual que una demostración matemática formal está basada en axiomas (las demostraciones geométricas están basadas a menudo en la geometría euclídea, mientras que otras demostraciones están basadas en los axiomas de la teoría de conjuntos), una demostración formal de la corrección de un programa está basada en las especificaciones con las que se ha diseñado el programa. Para demostrar que un programa ordena correctamente listas de nombres, podemos partir de la suposición de que la entrada del programa es una lista de nombres o si el programa está diseñado para calcular el promedio de uno o más números positivos, podemos suponer que la entrada consiste efectivamente en uno o más números positivos. En resumen, una demostración de corrección se inicia con la suposición de que hay una serie de condiciones, denominadas **precondiciones**, que se satisfacen al principio de la ejecución del programa.

El siguiente paso en una prueba de corrección consiste en considerar cómo se propagan a través del programa las consecuencias de estas precondiciones. Con este objetivo, los investigadores han analizado diversas estructuras de programa con el fin de determinar cómo se ve afectado por la ejecución de la estructura una afirmación que se sabe que es cierta antes de que se ejecute la estructura. Veamos un ejemplo sencillo. Si sabemos que una afirmación acerca del valor de `Y` es cierta antes de ejecutar la sentencia

```
X ← Y
```

entonces podemos hacer ese mismo enunciado acerca de `X` después de haber ejecutado la sentencia. Para ser más precisos, si el valor de `Y` es distinto de 0 antes de ejecutar la sentencia, entonces podemos concluir que el valor de `X` no será igual a 0 después de que la sentencia se ejecute.

Un ejemplo algo más complejo es el que se produce en el caso de una estructura **`if-then-else`** tal como

```
if (condición) then (sentencia A)
               else (sentencia B)
```

Aquí, si se sabe que una afirmación es cierta antes de ejecutar la estructura, entonces inmediatamente antes de ejecutar la `sentencia A`, sabremos que son ciertas tanto la afirmación como la condición que se ha comprobado, mientras que si hay que ejecutar la `sentencia B`, tendrán que ser ciertos tanto la afirmación como la negación de esa condición.

Más allá de la verificación del software

Los problemas de verificación, como se explica en el texto, no son exclusivos del software. Igualmente importante es el problema de confirmar que el hardware que ejecuta el programa está libre de errores. Esto implica la verificación de los diseños de los circuitos, así como de la construcción de la máquina. De nuevo, el estado actual de los conocimientos confía en gran medida en las pruebas lo que, como sucede con el software, implica que algunos errores muy sutiles pueden llegar a afectar a los productos acabados. La historia nos dice que el Mark I, construido en la Universidad de Harvard en la década de 1940, contenía errores de cableado que no fueron detectados durante muchos años. Un ejemplo más reciente es un fallo en la parte de coma flotante de los primeros microprocesadores Pentium. En ambos casos, el error fue detectado antes de que se produjera ninguna consecuencia grave.

Siguiendo reglas como estas, una demostración de corrección se realiza identificando afirmaciones, denominads **aserciones**, que pueden establecerse en diversos puntos del programa. El resultado es un conjunto de aserciones, cada una de las cuales es una consecuencia de las precondiciones del programa y de la secuencia de sentencias que conducen al punto del programa en el que se realiza la aserción. Si la aserción establecida de este modo al final del programa se corresponde con las especificaciones de salida deseadas (que se denominan **postcondiciones**), podemos concluir que el programa es correcto.

Por ejemplo, considere la típica estructura de bucle `while` representada en la Figura 5.23. Suponga que, como consecuencia de las precondiciones proporcionadas en el punto A, podemos establecer que una aserción concreta es verdadera cada vez que se lleva a cabo la comprobación de finalización (punto B) durante el proceso repetitivo. (Una aserción en un punto de un bucle que es cierta cada vez que se alcanza dicho punto del bucle se conoce con el nombre de **invariante de bucle**.) Entonces, si la secuencia de repetición llega a terminar alguna vez, la ejecución se moverá al punto C, en el que podremos concluir que se cumplen tanto el invariante de bucle como la condición de terminación. (El invariante de bucle sigue cumpliéndose porque la prueba de terminación no altera ningún valor del programa y la condición de terminación se cumple porque en caso contrario el bucle no terminaría.) Si estos enunciados combinados implican que son ciertas las postcondiciones deseadas, podemos completar nuestra demostración de corrección simplemente demostrando que las componentes de inicialización y de modificación del bucle terminan por conducir a la condición de terminación.

Trate de comparar este análisis con el ejemplo de ordenación por inserción ilustrado en la Figura 5.11. El bucle externo en dicho programa está basado en el invariante de bucle

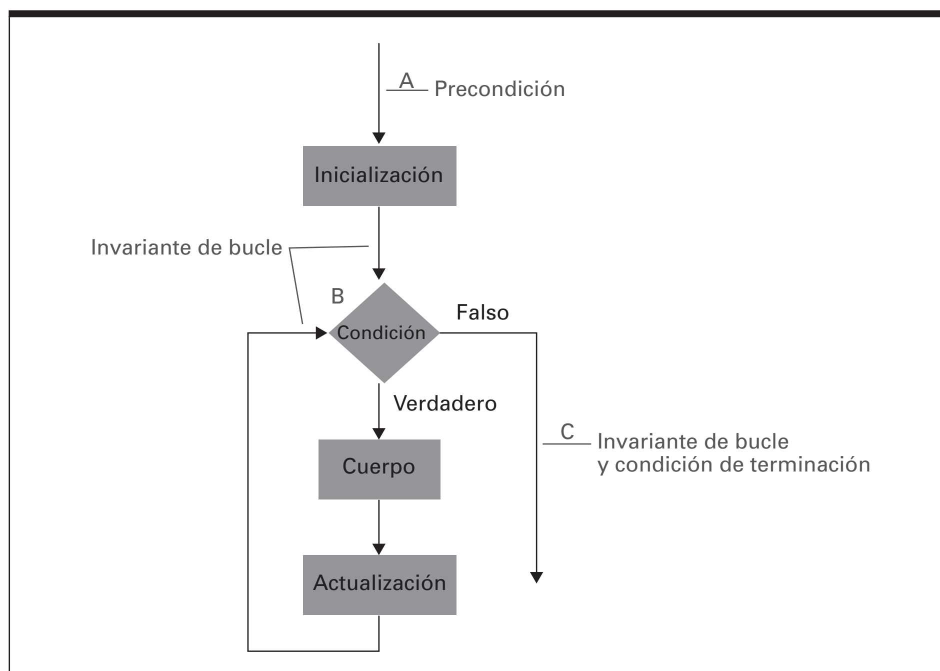
> Cada vez que se realice la comprobación de terminación, las entradas de la lista situadas en las posiciones 1 a $N - 1$ estarán ordenadas

y la condición de terminación es

> El valor de N es mayor que la longitud de la lista

Por tanto, si el bucle llega a terminar alguna vez, sabemos que ambas condiciones deben satisfacerse, lo que implica que la lista completa estará ordenada.

Figura 5.23 Las aserciones asociadas con una estructura while típica.



El progreso en el desarrollo de técnicas de verificación de programas sigue siendo muy complejo. Sin embargo, se están haciendo avances. Uno de los más significativos es el representado por el lenguaje de programación SPARK, que está estrechamente relacionado con otro lenguaje más popular, el Ada (Ada es uno de los lenguajes del que extraeremos ejemplos en el siguiente capítulo). Además de permitir expresar los programas en un formato de alto nivel similar a nuestro pseudocódigo, SPARK proporciona a los programadores un medio de incluir aserciones tales como precondiciones, postcondiciones e invariantes de bucle dentro del programa. De ese modo, un programa escrito en SPARK no solo contiene el algoritmo que hay que aplicar, sino también la información requerida para la aplicación de técnicas formales que permitan probar la corrección del programa. Hasta la fecha, SPARK se ha utilizado con éxito en numerosos proyectos de desarrollo software que incluyen aplicaciones software críticas, entre las que se encuentran aplicaciones de software seguro para la Agencia Nacional de Seguridad de Estados Unidos, el software de control interno utilizado en los aviones Hércules C130J de Lockheed Martin y sistemas críticos de control de transporte ferroviario.

A pesar de éxitos tales como SPARK, la utilización de técnicas formales de verificación de programas todavía no se ha generalizado, por lo que la mayor parte del software actual se "verifica" mediante pruebas, un proceso que dista mucho de proporcionar verdaderas garantías de corrección. Después de todo, la verificación mediante pruebas solo demuestra que el programa funciona correctamente en las circunstancias bajo las que ha sido probado. Cualquier conclusión adicional es una mera extrapolación. Los errores contenidos en un programa suelen ser consecuencia de sutiles despistes, de detalles que se pasan

fácilmente por alto tanto durante el desarrollo como durante las pruebas. En consecuencia, los errores de un programa, al igual que nuestro error en la cadena de oro, pueden no ser detectados, y a menudo sucede así, aunque se hayan hecho esfuerzos significativos para evitar que eso suceda. Un ejemplo catastrófico fue el que tuvo lugar en AT&T: un error en el software encargado del control de las centrales telefónicas modelo 114 no fue detectado desde que se instaló el software en diciembre de 1989 hasta el 15 de enero de 1990, en cuyo momento un conjunto muy particular de circunstancias provocó que unos cinco millones de llamadas quedaran innecesariamente bloqueadas durante un periodo de nueve horas.

Cuestiones y ejercicios

1. Suponga que nos encontramos con que una máquina programada con nuestro algoritmo de ordenación por inserción requiere un promedio de un segundo para ordenar una lista de 100 nombres. ¿Cuánto tiempo estima que se necesitará para ordenar una lista de 1000 nombres? ¿Y si la lista contiene 10.000 nombres?
2. Proporcione un ejemplo de un algoritmo perteneciente a cada una de las siguientes clases: $\Theta(lg\ n)$, $\Theta(n)$ y $\Theta(n^2)$.
3. Enumere las clases $\Theta(n^2)$, $\Theta(lg\ n)$, $\Theta(n)$ y $\Theta(n^3)$ por orden decreciente de eficiencia.
4. Considere el siguiente problema y la respuesta propuesta. ¿Es correcta la respuesta propuesta? ¿Por qué?

   Problema: Suponga que una caja contiene tres cartas. Una de las tres cartas está pintada de negro por ambas caras, otra está pintada de rojo por ambas caras y la tercera está pintada de rojo por una cara y de negro por la otra. Extraemos una de las cartas de la caja y se nos permite que veamos solo uno de los lados de la misma. ¿Cuál es la probabilidad de que el otro lado de la carta sea del mismo color que el lado que estamos viendo?
   Respuesta propuesta: Un medio (1/2). Suponga que el lado de la carta que puede ver es rojo (el argumento sería simétrico si fuera negro). Solo dos cartas de las tres tienen un lado rojo. Por tanto, la carta que vemos tiene que ser una de estas dos. Una de esas dos cartas es de color rojo por el otro lado, mientras que la otra es de color negro. Por tanto, es igualmente probable que la carta que vemos sea de color rojo por el otro lado que el que sea de color negro.

5. El siguiente fragmento de código es un intento de calcular el cociente (olvidándose de cualquier resto) de dos enteros positivos (un dividendo y un divisor), contando el número de veces que puede restarse el divisor del dividendo antes de que el valor restante sea menor que el divisor. Por ejemplo, $7/3$ nos daría 2, porque podemos restar el valor 3 del valor 7 dos veces. ¿Es correcto el programa? Justifique su respuesta.

```
Contador ← 0;
Resto ← Dividendo;
repeat (Resto ← Resto - Divisor;
        Contador ← Contador + 1)
until (Resto < Divisor)
Cociente ← Contador.
```

6. El siguiente fragmento de código está diseñado para calcular el producto de dos enteros no negativos `X` e `Y` acumulando la suma de `X` copias de `Y`; es decir, 3 por 4 se calcula acumulando la suma de tres cuatros. ¿Es correcto el programa? Justifique su respuesta.

```
Producto ← Y;
Contador ← 1;
while (Contador < X) do
 (Producto ← Producto + Y;
  Contador ← Contador + 1)
```

7. Asumiendo la precondición de que el valor asociado con N es un entero positivo, establezca un invariante de bucle que lleve a la conclusión de que si la siguiente rutina termina, entonces `Suma` tendrá asignado el valor $0 + 1 + \ldots + N$.

```
Suma ← 0;
K ← 0;
while (K < N) do
 (K ← K + 1;
  Suma ← Suma + K)
```

Proporcione un argumento que demuestre que esta rutina en efecto termina.

8. Suponga que tanto un programa como el hardware que lo ejecuta han sido formalmente verificados para garantizar su precisión. ¿Garantiza esto la precisión del sistema completo?

Problemas de repaso

(Los problemas con asterisco están asociados con las secciones opcionales)

1. Proporcione un ejemplo de un conjunto de pasos que se ajuste a la definición informal de algoritmo dada en el primer párrafo de la Sección 5.1, pero que no se ajuste a la definición formal dada en la Figura 5.1.
2. Defina algoritmo. Cite las distintas fases de la técnica de resolución de problemas en el contexto del desarrollo de software.
3. Explique la diferencia entre las definiciones formal e informal de algoritmo utilizando ejemplos adecuados.
4. Seleccione un tema con el que esté familiarizado y diseñe un pseudocódigo para proporcionar instrucciones relativas a dicho tema. En concreto, describa las primitivas que utilizaría y la sintaxis que emplearía

para representarlas. (Si tiene problemas en elegir un tema, inténtelo con los deportes, las artes o las manualidades.)

5. ¿Representa el siguiente programa un algoritmo en sentido estricto? ¿Por qué?

```
Var ← 0;
while (Var distinto de 10) do
 (Var ← Var + 3)
```

6. ¿En qué sentido los siguientes tres pasos no constituyen un algoritmo?

Paso 1: Dibujar un segmento de línea recta entre los puntos de coordenadas rectangulares (2,5) y (6,11).

Paso 2: Dibujar un segmento de línea recta entre los puntos de coordenadas rectangulares (1,3) y (3,6).

Paso 3: Dibujar un círculo cuyo centro se encuentre en la intersección de los dos segmentos de línea anteriores y cuyo radio sea igual a dos.

7. Escriba de nuevo el siguiente fragmento de código utilizando un bucle `repeat` en lugar de un bucle `while`. Asegúrese de que la nueva versión imprima los mismos valores que la original.

```
Var ← 2;
while (Var < 10) do
 (imprimir el valor asignado a Var y
  Var ← Var + 1)
```

8. Escriba el siguiente fragmento de código utilizando un bucle `while` en lugar de un bucle `repeat`. Asegúrese de que la nueva versión imprima los mismos valores que la original.

```
Var ← 2;
repeat
 (imprimir el valor asignado a Var y
  Var ← Var + 2)
until (Var = 10)
```

9. Indique qué habría que hacer para traducir un bucle de comprobación a la salida expresado en la forma

```
repeat (. . .) until (. . .)
```

a un bucle de de comprobación a la salida equivalente expresado en la forma

```
do (. . .) while (. . .)
```

10. Diseñe un algoritmo que, al proporcionarle una combinación de los dígitos 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, reordene los dígitos de modo que la nueva ordenación represente el siguiente número más grande que pueda representarse mediante dichos dígitos (o que informe de que no existe dicha reordenación en caso de que no haya ninguna que genere un valor mayor). Así, 5647382901 nos daría 5647382910.

11. Diseñe un algoritmo para encontrar todos los factores de un entero positivo. Por ejemplo, en el caso del entero 12, el algoritmo debe devolver los valores 1, 2, 3, 4, 6 y 12.

12. Explique las diferencias que existen entre la iteración y la recursión.

13. ¿Cuál es la diferencia entre la metodología de arriba-abajo y la metodología de abajo-arriba?

14. ¿Cuál es la diferencia entre precondiciones y postcondiciones?

15. El siguiente es un problema de suma en notación base diez tradicional. Cada letra representa un dígito diferente. ¿Qué dígito representa cada letra? ¿Cómo ha logrado abrirse camino hacia la solución del problema?

```
  XYZ
+ YWY
 ----
 ZYZW
```

16. El siguiente es un problema de multiplicación en notación base diez tradicional. Cada letra representa un dígito diferente. ¿Qué dígito representa cada letra? ¿Cómo ha logrado abrirse camino hacia la solución del problema?

```
   XY
 × YX
 ----
   XY
  YZ
 ----
  WVY
```

17. El siguiente es un problema de suma en notación binaria. Cada letra representa un dígito binario distinto. ¿Qué letra representa el 1 y qué letra representa el 0? Diseñe un algoritmo para resolver problemas como este.

```
   YXX
+  XYX
  XYYY
```

18. Cuatro prospectores que solo disponen de una linterna deben atravesar la galería de una mina. Como máximo, solo dos prospectores pueden caminar juntos y cualquier prospector que trate de atravesar la galería debe llevar consigo la linterna. Los prospectores, cuyos nombres son Andrés, Benito, Juan y Kety, pueden atravesar la galería en uno, dos, cuatro y ocho minutos, respectivamente. Cuando dos caminan juntos, se desplazan a la velocidad del más lento de los dos. ¿Cómo pueden los cuatro prospectores cruzar la galería en solo 15 minutos? Después de haber resuelto este problema, explique cómo ha conseguido abrir brecha en él.

19. Comenzando con un taza grande y otra pequeña, llene la taza pequeña con té y luego viértalo en la taza de mayor tamaño. A continuación, rellene la taza pequeña con leche y eche parte de esa leche en la taza más grande. Mezcle el contenido de la taza grande y luego vierta la mezcla en la taza pequeña hasta que esté llena. ¿Habrá más leche en el taza grande que té en la taza pequeña? Después de haber resuelto este problema, explique cómo ha conseguido abrir brecha en él.

20. Dos abejas, de nombres Romeo y Julieta, viven en diferentes colmenas pero se encuentran accidentalmente y se enamoran. En una mañana de primavera sin viento abandonan simultáneamente sus respectivas colmenas para visitarse. Sus rutas se encuentran en un punto situado a 50 metros de la colmena más próxima, pero desafortunadamente no se ven y continúan hasta su destino. Al llegar a él, invierten la misma cantidad de tiempo en descubrir que el otro no se encuentra en casa y comienzan su viaje de vuelta. En el viaje de vuelta se encuentran en un punto que está a 20 metros de la colmena más próxima. Ambos se ven y se van a comer de picnic antes de volver a casa. ¿A qué distancia están las dos colmenas? Después de haber resuelto este problema, explique cómo ha conseguido abrir brecha en él.

21. Diseñe un algoritmo que, dadas dos cadenas de caracteres, compruebe si la primera cadena aparece como subcadena en algún punto de la segunda cadena.

22. El siguiente algoritmo está diseñado para imprimir el principio de lo que se conoce como la secuencia de Fibonacci. Identifique el cuerpo del bucle. ¿Dónde está el paso de inicialización para el control del bucle? ¿Y el paso de actualización? ¿Y el paso de comprobación? ¿Qué lista de números se generará?

```
Último ← 0;
ValorActual ← 1;
while (ValorActual < 50) do
(imprimir el valor asignado a ValorActual;
  Temp ← Último;
  Último ← ValorActual; y
  ValorActual ← Último + Temp)
```

23. ¿Qué secuencia de números imprimirá el siguiente algoritmo si se inicia con los valores de entrada 0 y 1?

```
procedure ImpresionMisteriosa (
  Último, Actual)
if (Actual < 100) then
(imprimir el valor asignado a Actual;
 Temp ← Actual + Último;
 aplicar ImpresionMisteriosa a los
   valores Actual y Temp)
```

24. Modifique el procedimiento `ImpresionMisteriosa` del problema anterior para que los valores se impriman en orden inverso.

25. ¿Qué letras se comprobarán en el algoritmo de búsqueda binaria (Figura 5.14) si se aplica a la lista A, B, C, D, E, F, G, H, I, J, K, L, M, N, O mientras se está buscando el valor J? ¿Y qué ocurre si se está buscando el valor Z?

26. Después de realizar muchas búsquedas secuenciales en una lista de 6.000 entradas, ¿cuál esperaría que fuera el número medio de veces que se ha comparado el valor objetivo con una entrada de la lista? ¿Y cuál sería ese número medio de veces si el algo-

ritmo de búsqueda fuera el de búsqueda binaria?

27. Identifique la condición de terminación en cada una de las siguientes sentencias iterativas.

a. `while (Var < 10) do ( )`

b.
```
repeat (            )
  until (Var = 5)
```

c.
```
while (( Var < 10) y (Suma < 30))
do (            )
```

28. Identifique el cuerpo de la siguiente estructura de bucle y cuente el número de veces que se ejecutará. ¿Qué sucede si cambiamos la comprobación por "`while` (Var distinto de 8) do"?

```
Var ← 1;
while (Var distinto de 5) do
  (imprimir el valor asignado a Var y
   Var ← Var + 2)
```

29. ¿Qué problemas esperaría que surjan si se implementara el siguiente programa en una computadora? (*Consejo:* recuerde el problema de los errores de redondeo asociados con la aritmética en punto flotante.)

```
Cuenta ← 0.1;
repeat
  (imprimir el valor asignado a
   Cuenta y
   Cuenta ← Cuenta + 0.1)
until (Cuenta igual a 1)
```

30. Diseñe una versión recursiva del algoritmo de Euclides (Cuestión 3 de la Sección 5.2).

31. Suponga que aplicamos tanto `Test1` como `Test2` (definidos a continuación) al valor de entrada 1. ¿Cuál será la diferencia entre la salida impresa de las dos rutinas?

```
procedure Test1 (Cuenta)
if (Cuenta distinta de 5)
  then (imprimir el valor asignado a
         Cuenta;
        aplicar Test1 al valor
         Cuenta + 1)

procedure Test2 ( Cuenta)
if (Cuenta distinta de 5)
  then (aplicar Test2 al valor
         Cuenta + 1;
        imprimir el valor asignado a
         Cuenta)
```

32. Identifique los componentes importantes del mecanismo de control en las rutinas del problema anterior. En particular, ¿qué condición hace que el proceso termine? ¿Dónde se modifica el estado del proceso para intentar acercarse a esa condición de terminación? ¿Dónde se inicializa el proceso de control?

33. Identifique la condición de finalización en el siguiente procedimiento recursivo.

```
procedure ABC (N)
if (NUM = 10) then (aplicar el
                    procedimiento ABC
                    al valor NUM + 1)
```

34. Aplique el procedimiento `ImpresionMisteriosa` (definido a continuación) al valor 3 y anote los valores que se imprimen.

```
procedure ImpresionMisteriosa (N)
if (N > 0) then (imprimir el valor de
                 de N y aplicar el
                 procedimiento
                 ImpresionMisteriosa
                 al valor N - 2)
Imprimir el valor de N + 1.
```

35. Aplique el procedimiento `ImpresionMisteriosa` (definido a continuación) al valor 2 y anote los valores que se imprimen.

```
procedure ImpresionMisteriosa (N)
if (N > 0)
  then (imprimir el valor de N y
        aplicar el procedimiento
        ImpresionMisteriosa
         al valor N - 2)
  else (imprimir el valor de N y
         if (N > -1)
           then (aplicar el
                 procedimiento
                 ImpresionMisteriosa
                 al valor N + 1))
```

36. Diseñe un algoritmo para generar la secuencia de enteros positivos (en orden creciente) cuyos únicos divisores primos sean 2 y 3; es decir, el programa debe generar la secuencia 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 27,... ¿Representa el programa un algoritmo en sentido estricto?

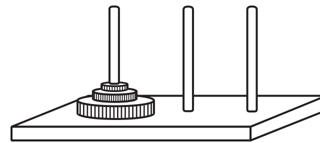
37. Responda a las siguientes cuestiones en función de la lista Alicia, Benito, Carol, Diana, Elena, Flora, Gerardo, Héctor, Irene.

a. ¿Qué algoritmo de búsqueda (secuencial o binaria) encontraría el nombre de Gerardo más rápidamente?
b. ¿Qué algoritmo de búsqueda (secuencial o binaria) encontraría el nombre de Alicia más rápidamente?
c. ¿Qué algoritmo de búsqueda (secuencial o binaria) detectaría la ausencia del nombre Bruno más rápidamente?
d. ¿Qué algoritmo de búsqueda (secuencial o binaria) detectaría la ausencia del nombre Susana más rápidamente?
e. ¿Cuántas entradas se comprobarán al buscar el nombre Elena si usamos la búsqueda secuencial? ¿Cuántas se comprobarían al utilizar la búsqueda binaria?

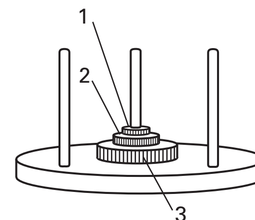
38. El factorial de 0 es por definición igual a 1. El factorial de un entero positivo se define como el producto de dicho entero por el factorial del siguiente entero más pequeño no negativo. Utilizamos la notación `n!` para expresar el factorial del entero `n`. Por tanto, el factorial de 3 (que se escribe 3!) es $3 \times (2!) = 3 \times (2 \times (1!)) = 3 \times (2 \times (1 \times (0!))) = 3 \times (2 \times (1 \times (1))) = 6$. Diseñe un algoritmo recursivo que calcule el factorial de un valor dado.

39. a. Suponga que tiene que ordenar una lista de cinco nombres y que ya ha diseñado un algoritmo que ordena una lista de cuatro nombres. Diseñe un algoritmo para ordenar la lista de cinco nombres aprovechando el algoritmo previamente diseñado.
b. Diseñe un algoritmo recursivo para ordenar listas arbitrarias de nombres, basándose en la técnica utilizada en el apartado (a).

40. El puzzle denominado las Torres de Hanoi está compuesto por tres palos, uno de los cuales contiene varios anillos, apilados por orden de diámetro descendente de abajo a arriba. El problema consiste en mover la pila de anillos a otro de los palos. Solo se permite mover un anillo cada vez y nunca puede ponerse un anillo sobre otro que tenga un menor diámetro. Observe que si el puzzle solo incorporara un anillo sería extremadamente fácil. Además, al intentar solucionar el problema de mover varios anillos, vemos que si pudiéramos moverlos todos excepto el de mayor tamaño a otro palo, entonces el anillo de mayor tamaño podría colocarse en el tercer palo y el problema quedaría reducido a situar los restantes anillos encima de él. Teniendo en cuenta esta observación, desarrolle un algoritmo recursivo para resolver el puzzle de las Torres de Hanoi para un número arbitrario de anillos.



41. Otra técnica para resolver el puzzle de la Torres de Hanoi (Problema 40) consiste en imaginar que los palos están dispuestos de manera circular, con un palo montado en cada una de las posiciones correspondientes a las 4, las 8 y las 12 en punto. Los anillos, que comienzan en uno de los palos, están numerados 1, 2, 3, etc., siendo el anillo más pequeño el que tiene el número 1. Los anillos con numeración impar, cuando se encuentran en la parte superior de una pila, pueden moverse en el sentido de las agujas del reloj al palo siguiente; de la misma forma, los anillos con numeración par pueden moverse en sentido contrario a las agujas del reloj (siempre que dicho movimiento no coloque un anillo sobre otro más pequeño). Con esta restricción, trate siempre de mover el anillo con el número mayor de aquellos que puedan moverse. Basándose en esta observación, desarrolle un algoritmo no recursivo para resolver el puzzle de las Torres de Hanoi.



42. Desarrolle dos algoritmos, uno basado en una estructura de bucle y otro en una estructura recursiva, que impriman el salario diario de un trabajador al que cada día se le paga el doble del salario del día anterior (comenzando con un céntimo para el primer día de trabajo). Imprima los salarios diarios para un periodo de 30 días. ¿Qué problemas relativos al almacenamiento de números nos vamos con toda probabilidad a encontrar si tratamos de implementar esas soluciones en una máquina real?

43. Diseñe un algoritmo para calcular la raíz cuadrada de un número positivo comenzando con el propio número como primera aproximación y generando repetidamente una nueva aproximación a partir de la anterior, promediando la aproximación anterior con el resultado de dividir el número original entre la aproximación anterior. Analice el control de este proceso repetitivo. En particular, ¿qué condición debe hacer que terminen las repeticiones?

44. Diseñe un algoritmo que enumere todas las posibles reordenaciones de los símbolos de una cadena de cuatro caracteres distintos.

45. Diseñe un algoritmo que, a partir de una lista de nombres, determine el nombre más largo de la lista. Indique lo que hace su solución si hay varios nombres con la mayor longitud en la lista. En particular, ¿qué haría su algoritmo si todos los nombres tuvieran la misma longitud?

46. Diseñe un algoritmo que, a partir de una lista de cinco o más números, determine los cinco más pequeños y los cinco mayores de la lista sin ordenar la lista completa.

47. Ordene los nombres Fredo, Ricardo, Diana, Mateo, Sara y William en un orden que requiera el menor número de comparaciones al ordenar la lista mediante el algoritmo de ordenación por inserción.

48. ¿Cuál es el mayor número de entradas que habrá que comprobar si se aplica el algoritmo de búsqueda binaria (Figura 5.14) a una lista de 4.000 nombres? ¿ Cómo se compara ese valor con el correspondiente a la búsqueda secuencial (Figura 5.6)?

49. Utilice la notación zeta-mayúscula para clasificar los algoritmos tradicionales de suma y multiplicación que se enseñan en la escuela. Es decir, si nos piden sumar dos números, cada uno de los cuales tiene n dígitos, ¿cuántas sumas individuales habrá que realizar? Si nos piden multiplicar dos números de n dígitos, ¿cuántas multiplicaciones individuales harán falta?

50. En ocasiones, un pequeño cambio en un problema puede alterar significativamente la forma de su solución. Por ejemplo, determine un algoritmo simple para resolver el siguiente problema y clasifíquelo utilizando la notación zeta-mayúscula:

Dividir un grupo de personas en dos subgrupos disjuntos (de tamaño arbitrario), de modo que la diferencia en la edad total de los miembros de los dos subgrupos sea lo mayor posible.

Ahora cambie el problema de modo que la diferencia deseada sea lo más pequeña posible e indique a qué clase de problemas pertenece la nueva solución.

51. Extraiga de la siguiente lista un conjunto de números cuya suma sea igual a 3165. ¿Cuál es la eficiencia de su técnica de resolución de este problema?

26, 39, 104, 195, 403, 504, 793, 995, 1156, 1677

52. Examine el bucle de la siguiente rutina y diga si termina. Explique su respuesta. Explique lo que podría suceder si esta rutina fuera realmente ejecutada por una computadora (consulte la Sección 1.7).

```
I ← 1;
J ← 1/3;
while (I distinto de 0) do
   (I ← I – J;
    J ← J ÷ 3)
```

53. El siguiente fragmento de código está diseñado para calcular el producto de dos enteros no negativos `I` y `J` acumulando la suma de `I` copias de `J`; es decir, 3 por 4 se calcula acumulando la suma de tres cuatros. ¿Es correcto este segmento de programa? Explique su respuesta.

```
Resultado ← 0;
Cuenta ← 0;
repeat (Resultado ← Resultado + J,
        Cuenta ← Cuenta + 1)
until (Cuenta = I)
```

54. El siguiente fragmento de código de programa está diseñado para indicar cuál de los enteros positivos I y J es mayor. ¿Es correcto el fragmento de código de programa? Explique su respuesta.

```
Diferencia ← I – J;
if (Diferencia es positiva)
  then (imprimir "J es menor que I")
  else (imprimir "I es menor que J")
```

55. El siguiente fragmento de código de programa está diseñado para encontrar el mayor número de entre las entradas de una lista no vacía de enteros. ¿Es correcto? Explique su respuesta.

```
ValorAProbar ← primera entrada de la lista;
EntradaActual ← primera entrada de la lista;
while (EntradaActual no es la última
      entrada) do
  (if (EntradaActual > ValorAProbar)
       then (ValorAProbar ← EntradaActual)
  EntradaActual ← siguiente entrada de la
    lista)
```

56. a. Identifique las precondiciones para la búsqueda secuencial tal como se la representa en la Figura 5.6. Establezca un invariante de bucle para la estructura `while` de dicho programa que, al combinarse con la condición de finalización, implique que al terminar el bucle el algoritmo informe correctamente del éxito o del fallo de la búsqueda.

b. Proporcione un argumento que muestre que el bucle `while` de la Figura 5.6 sí que termina.

57. Basándose en las precondiciones de que a X e Y se les asignan enteros no negativos, identifique un invariante de bucle para la siguiente estructura `while` que, al combinarse con la condición de terminación, implique que el valor asociado con Z al terminar el bucle sea X – Y.

```
Z ← X;
J ← 0;
while (J < Y) do
   (Z ← Z – 1;
    J ← J + 1)
```

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Como actualmente es imposible verificar completamente la precisión de programas complejos, ¿en qué circunstancias, si es que hay alguna, debería ser considerado responsable de los errores el creador de un programa?

2. Suponga que ha tenido una idea y que la transforma en un producto que muchas personas pueden utilizar. Suponga además que esa idea ha requerido un año de trabajo y una inversión de 50.000 euros para desarrollarla en una forma que sea útil para el público en general. Sin embargo, en su forma final, el producto puede ser utilizado por la mayoría de las personas sin necesidad de comprarle nada a usted. ¿Qué derecho tiene a una compensación? ¿Es ético piratear el software de computadora? ¿Qué sucede con la música y las películas?

3. Suponga que un paquete software es tan caro que cae completamente fuera de su capacidad de compra. ¿Es ético copiarlo para uso particular? (Después de todo, no se está privando al suministrador de ninguna venta porque en ningún caso habría adquirido ese paquete software.)
4. La propiedad de los ríos, los bosques, los océanos, etc. ha sido durante mucho tiempo tema de debate. ¿En qué sentido debería concederse la propiedad de un algoritmo a alguna institución o persona?
5. Algunas personas creen que los nuevos algoritmos se descubren, mientras que otras piensan que los nuevos algoritmos se crean. ¿Con cuál de estas dos afirmaciones está más de acuerdo? ¿Conducirían los distintos puntos de vista a diferentes conclusiones en relación con la propiedad intelectual de los algoritmos y los derechos de propiedad?
6. ¿Es ético diseñar un algoritmo para llevar a cabo un acto ilegal? ¿Tiene alguna importancia que el algoritmo llegue o no a ser ejecutado alguna vez? ¿Debe la persona que crea ese algoritmo tener derechos de propiedad intelectual sobre el mismo? En caso afirmativo, ¿cuáles deberían ser esos derechos? ¿Deberían depender los derechos de propiedad del propósito de cada algoritmo? ¿Es ético divulgar y distribuir técnicas para romper la seguridad de un sistema? ¿Tiene alguna importancia el tipo de sistema que se quiera romper?
7. A un autor se le paga por los derechos de filmación de una novela aún cuando a menudo la historia se modifica en la película. ¿Cuánto debe cambiar una historia antes de transformarse en una historia diferente? ¿Qué alteraciones hay que hacer en un algoritmo para que se pueda considerar un algoritmo diferente?
8. En la actualidad se comercializa software educativo para niños de 18 meses de edad o menos. Los proponentes argumentan que dicho software proporciona imágenes y sonidos que de otro modo no estarían a disposición de muchos niños. Los oponentes argumentan que se trata de un mal sustituto de la interacción personal entre padres e hijos. ¿Cuál es su opinión? ¿Cree que debería tomar algún tipo de acción basándose en su opinión, sin conocer más detalles acerca del software? En caso afirmativo, ¿qué acciones cree que debería emprender?

Lecturas adicionales

Aho, A. V., J. E. Hopcroft y J. D. Ullman. ***The Design and Analysis of Computer Algorithms.*** Boston, MA: Addison-Wesley, 1974.

Baase, S. ***Computer Algorithms: Introduction to Design and Analysis***, 3ª ed. Boston, MA: Addison-Wesley, 2000.

Barnes, J. ***High Integrity Software: The SPARK Approach to Safety and Security.*** Boston, MA: Addison-Wesley, 2003.

Gries, D. ***The Science of Programming***. Nueva York: Springer-Verlag, 1998.

Harbin, R. ***Origami—the Art of Paper Folding***. Londres: Hodder Paperbacks, 1973.

Johnsonbaugh, R. y M. Schaefer. ***Algorithms***. Upper Saddle River, NJ: Prentice-Hall, 2004.

Kleinberg, J. y E. Tardos. ***Algorithm Design***. Boston, MA: Addision-Wesley, 2006.

Knuth, D. E. ***The Art of Computer Programming***, Vol. 3, 3ª ed. Boston, MA: Addison-Wesley, 1998.

Levitin, A. V. ***Introduction to the Design and Analysis of Algorithms***, 2ª ed. Boston, MA: Addison-Wesley, 2007.

Polya, G. ***How to Solve It***. Princeton, NJ: Princeton University Press, 1973.

Roberts, E. S. ***Thinking Recursively***. Nueva York: Wiley, 1986.

CAPÍTULO

6

Lenguajes de programación

En este capítulo vamos a estudiar los lenguajes de programación. Nuestro propósito no es aprender ningún lenguaje concreto, sino más bien aprender *acerca* de los lenguajes de programación. De lo que se trata es de reconocer los aspectos comunes y las diferencias entre los distintos lenguajes de programación y sus metodologías asociadas.

Las secciones marcadas con asterisco se sugieren como secciones opcionales.

El desarrollo de sistemas complejos de software como por ejemplo sistemas operativos, software de red y el amplio rango del software de aplicación disponible hoy día, sería probablemente imposible si los seres humanos nos viéramos forzados a escribir los programas en lenguaje máquina. El tratar con los intrincados detalles asociados con dichos lenguajes al mismo tiempo que se intenta organizar sistemas complejos sería una experiencia aterradora, por decirlo suavemente. En consecuencia, se han desarrollado lenguajes de programación similares a nuestro pseudocódigo que permiten expresar los algoritmos en una forma que es tanto aceptable para los seres humanos como fácilmente convertible en instrucciones de lenguaje máquina. Nuestro objetivo en este capítulo es explorar esa esfera de las Ciencias de la computación que trata del diseño e implementación de dichos lenguajes.

6.1 Perspectiva histórica

Comencemos nuestro estudio trazando el desarrollo histórico de los lenguajes de programación.

Primeras generaciones

Como hemos estudiado en el Capítulo 2, los programas para las computadoras modernas constan de secuencias de instrucciones codificadas como dígitos numéricos. Tal sistema de codificación se conoce como lenguaje máquina. Lamentablemente, escribir programas en lenguaje máquina es una tarea tediosa que a menudo conduce a errores que se deben localizar y corregir (proceso que se conoce como **depuración**) antes de dar la tarea por finalizada.

En la década de 1940, los investigadores simplificaron el proceso de programación desarrollando sistemas de notación que permitían representar las instrucciones en forma mnemónica, en lugar de en forma numérica. Por ejemplo, la instrucción

Mover el contenido del registro 5 al registro 6

se expresaría como

```
4056
```

utilizando el lenguaje máquina presentado en el Capítulo 2, mientras que un sistema mnemónico se expresaría como

```
MOV R5, R6
```

Veamos otro ejemplo más extenso. La rutina en lenguaje máquina

```
156C
166D
5056
306E
C000
```

que suma el contenido de las celdas de memoria 6C y 6D y almacena el resultado en la posición 6E (Figura 2.7 del Capítulo 2), utilizando mnemónicos se expresaría como

```
LD R5,Precio
```

```
LD R6,CosteTransporte
ADDI R0,R5 R6
ST R0,CosteTotal
HLT
```

(En este caso hemos utilizado LD, ADDI, ST y HLT para representar las instrucciones ***load*** (cargar), ***add*** (sumar), ***store*** (almacenar) y ***halt*** (parar). Además, hemos utilizado nombres descriptivos `Precio`, `CosteTransporte` y `CosteTotal` para hacer referencia a las celdas de memoria situadas en las posiciones 6C, 6D y 6E, respectivamente. Estos nombres descriptivos suelen denominarse **identificadores**. Observe que la forma mnemónica, aunque dista mucho de ser perfecta, representa mucho mejor que la forma numérica el significado de la rutina.

Una vez que se establecieron este tipo de sistemas mnemónicos, se desarrollaron programas conocidos como **ensambladores** para convertir las expresiones mnemónicas en instrucciones de lenguaje máquina. De este modo, en lugar de verse obligados a desarrollar un programa directamente en lenguaje máquina, se podría desarrollar un programa en forma mnemónica y luego convertirlo a lenguaje máquina por medio de un ensamblador.

Un sistema mnemónico para la representación de programas se denomina **lenguaje ensamblador**. En la época en la que se desarrollaron los lenguajes ensambladores representaron un gran paso en la búsqueda de mejores técnicas de programación. De hecho, los lenguajes ensambladores fueron tan revolucionarios que se los denominó lenguajes de segunda generación, siendo la primera generación los propios lenguajes máquina.

Aunque los lenguajes ensambladores tienen muchas ventajas con respecto a sus correspondientes lenguajes máquina, siguen proporcionando un entorno de programación con múltiples carencias. Después de todo, las primitivas utilizadas en un lenguaje ensamblador son esencialmente las mismas que podemos encontrar en el lenguaje máquina correspondiente. La diferencia estriba simplemente en la sintaxis utilizada para representarlas. Así, un programa escrito en un lenguaje ensamblador es inherentemente dependiente de la máquina; es decir, las instrucciones del programa se expresan en términos de las características de una máquina concreta. Como consecuencia, un programa escrito en lenguaje ensamblador no puede moverse fácilmente a otra computadora con una arquitectura diferente, porque es necesario reescribirlo para adaptarse a la configuración de registros y al conjunto de instrucciones de la nueva computadora.

Otra desventaja del lenguaje ensamblador es que un programador, aunque ya no está obligado a codificar las instrucciones en forma numérica, sigue viéndose forzado a pensar en términos de los pequeños pasos incrementales del lenguaje máquina. La situación es análoga a diseñar una vivienda en términos de tableros de madera, clavos, ladrillos, etc. Es cierto que la construcción real de la vivienda requerirá en un último término una descripción basada en esos componentes elementales, pero el proceso de diseño es más fácil si pensamos en términos de unidades de mayor tamaño como habitaciones, ventanas, puertas, etc.

En resumen, las primitivas elementales en las que un producto debe basarse en último término no tienen por qué ser necesariamente las primitivas que hay que emplear durante el diseño del producto. El proceso de diseño se adapta mejor al uso de primitivas de alto nivel, cada una de las cuales representa un concepto asociado con una de las características principales del pro-

ducto. Una vez completado el diseño, esas primitivas pueden traducirse a conceptos de menor nivel relacionados con los detalles de implementación.

Siguiendo esta filosofía, los expertos en Ciencias de la computación comenzaron a desarrollar lenguajes de programación que se adaptaran mejor al desarrollo software que los lenguajes ensambladores de bajo nivel. El resultado fue la aparición de una tercera generación de lenguajes de programación que difería de las generaciones anteriores en el sentido de que sus primitivas eran tanto de un mayor nivel (porque expresaban las instrucciones en incrementos de mayor tamaño) como **independientes de la máquina** (ya que no dependían de las características de ninguna máquina concreta). Los más conocidos de entre los primeros ejemplos de estos lenguajes son FORTRAN (*FORmula TRANslator*, Traductor de fórmulas), que fue desarrollado para aplicaciones científicas y de ingeniería y COBOL (*COmmon Business-Oriented Language*, Lenguaje común orientado a la empresa), que fue desarrollado por la Armada de Estados Unidos para aplicaciones empresariales.

En general, el enfoque adoptado con los lenguajes de programación de tercera generación era identificar un conjunto de primitivas de alto nivel (básicamente con el mismo espíritu con el que hemos desarrollado nuestro pseudocódigo del Capítulo 5) con las que pudiera desarrollarse software. Cada una de estas primitivas se desarrollaba de forma que pudiera implementarse mediante una secuencia de las primitivas de bajo nivel disponibles en los lenguajes máquina. Por ejemplo, la instrucción

```
assign CosteTotal the value Precio + CosteTransporte
```

expresa una actividad de alto nivel (asignar a una variable el resultado de una operación) sin hacer referencia a cómo una determinada máquina debería realizar la tarea, a pesar de lo cual podría ser implementada mediante la secuencia de instrucciones máquina vista anteriormente. Por tanto, nuestra estructura de pseudocódigo

```
identificador ← expresión
```

es, potencialmente, una primitiva de alto nivel.

Una vez identificado este conjunto de primitivas de alto nivel, se escribía un programa denominado **traductor**, que traducía a lenguaje máquina los programas expresados en estas primitivas de alto nivel. Dicho traductor era similar a los ensambladores de segunda generación, salvo porque a menudo tenía que compilar secuencias cortas de varias instrucciones máquina para simular la actividad solicitada por una única primitiva de alto nivel. Por ello, estos programas de traducción se les denominó **compiladores**.

Como alternativa a los traductores surgió otra manera de implementar los lenguajes de tercera generación, los denominados **intérpretes**. Estos programas eran similares a los traductores, salvo porque ejecutaban las instrucciones a medida que se iba traduciendo, en lugar de grabar la versión traducida para uso futuro. Es decir, en lugar de generar una copia en lenguaje máquina de un programa para ejecutarla posteriormente, el intérprete ejecutaba realmente el programa a partir de su formulación de alto nivel.

Como curiosidad histórica dejemos constancia de que la tarea de promover los lenguajes de programación de tercera generación no fue tan fácil como cabría imaginar. La idea de escribir programas en un lenguaje similar al len-

guaje natural era tan revolucionaria que muchas personas que ocupaban puestos de decisión se resistieron a esta idea al principio. Grace Hopper, a quien se le atribuye el desarrollo del primer compilador, contaba a menudo la historia de que en cierta ocasión en la que estaba mostrando un traductor para un lenguaje de tercera generación en el que se utilizaban términos en alemán en lugar de en inglés, con el objetivo de ilustrar que el lenguaje de programación estaba construido por un pequeño conjunto de primitivas que podían expresarse en distintos idiomas efectuando unas simples modificaciones al traductor, se vio sorprendida al comprobar que muchas personas de la audiencia se escandalizaron de que, en aquellos años próximos a la Segunda Guerra Mundial, ella estuviera enseñando a una computadora a "entender" el alemán. Hoy día sabemos que comprender un lenguaje natural implica muchísimo más que responder a unas pocas primitivas rigurosamente definidas. De hecho, los **lenguajes naturales** (como el inglés, el alemán, el español o el latín) se distinguen de los **lenguajes formales** (como los lenguajes de programación) en que estos últimos están definidos de forma muy precisa mediante gramáticas (Sección 6.4), mientras que los primeros han evolucionado a lo largo del tiempo sin un análisis gramatical formal.

Más allá de la independencia de la máquina

Con el desarrollo de los lenguajes de tercera generación, se consiguió en buena medida el objetivo de la independencia con respecto a la máquina. Puesto que las sentencias en un lenguaje de tercera generación no hacen referencia a las características de ninguna máquina concreta, pueden compilarse con la misma facilidad para un tipo de máquina que para otra. Un programa escrito en un lenguaje de tercera generación podía, en teoría, utilizarse en cualquier máquina simplemente compilándolo con el compilador apropiado.

Sin embargo, la realidad ha demostrado no ser tan simple. Cuando se diseña un compilador, las características particulares de la máquina subyacente se reflejan en ocasiones en forma de condiciones impuestas al lenguaje que se está traduciendo. Por ejemplo, las diferentes maneras en que las máqui-

Software multiplataforma

Un programa de aplicación típico depende del sistema operativo para realizar muchas de sus tareas. Puede necesitar los servicios del administrador de ventanas para comunicarse con el usuario de la computadora, o puede tener que usar el administrador de archivos para extraer datos del disco duro. Lamentablemente, los diferentes sistemas operativos obligan a que las solicitudes de estos servicios se realicen de formas distintas. Por tanto, para poder transferir y ejecutar programas a través de redes e interredes que impliquen distintas arquitecturas hardware y distintos sistemas operativos, los programas deben ser independientes del sistema operativo además de ser independientes de la máquina. Suele utilizarse el término multiplataforma para reflejar este nivel adicional de independencia. Es decir, el software multiplataforma es software que, además de ser independiente del diseño hardware de la máquina, también es independiente del sistema operativo y puede, por tanto, ejecutarse a través de una red.

nas gestionan las operaciones de E/S han provocado históricamente que el “mismo” lenguaje tenga diferentes características, o dialectos, en las distintas máquinas. En consecuencia, a menudo es necesario realizar modificaciones de carácter menor a un programa, antes de poder moverlo de una máquina a otra.

Para complicar aún más este problema de la portabilidad, existe una falta de acuerdo en algunos casos sobre cuál es la definición correcta de un lenguaje concreto. Para ayudar en este sentido, ANSI (American National Standards Institute) e ISO (International Organization for Standardization) han adoptado y publicado estándares para muchos de los lenguajes más populares. En otros casos, debido a la popularidad de un cierto dialecto de un lenguaje y al deseo de otros desarrolladores de compiladores de diseñar productos compatibles, se han dessarrollado estándares informales . Sin embargo, incluso en el caso de los lenguajes altamente estandarizados, los diseñadores de compiladores a menudo proporcionan funcionalidades, que en ocasiones se denominan extensiones del lenguaje, que no forman parte de la versión estándar del mismo. Si un programador aprovecha estas funcionalidades, el programa generado no será compatible con aquellos entornos en los se emplee un compilador de un fabricante diferente.

En la historia de los lenguajes de programación, el hecho de que los lenguajes de tercera generación no llegaran a proporcionar una verdadera independencia con respecto a la máquina tiene, en la práctica, poca importancia por dos razones distintas. En primer lugar, estaban tan cerca de ser auténticamemte independientes con respecto a la máquina que el software podía moverse de una máquina a otra con relativa facilidad. En segundo lugar, el objetivo de la independencia con respecto a la máquina resultó ser tan solo la semilla de otros objetivos mucho más ambiciosos. De hecho, el darse cuenta de que las máquinas podían responder a sentencias de alto nivel tales como

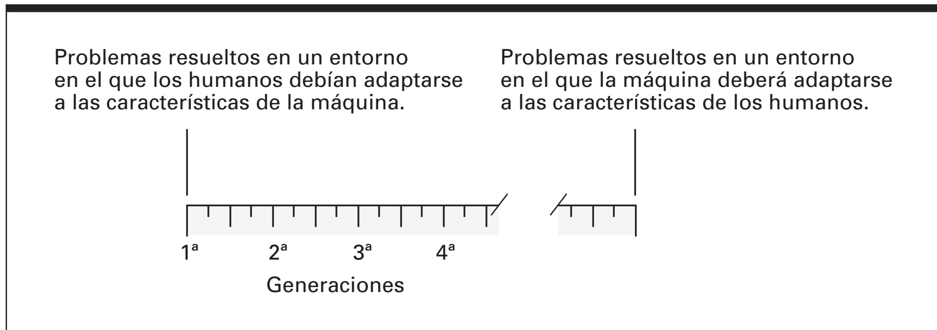
```
assign CosteTotal the value Precio + CosteTransporte
```

llevó a los expertos en computación a soñar con la creación de entornos de programación que permitieran a los humanos comunicarse con las máquinas en términos de conceptos abstractos en lugar de forzar a las personas a traducir esos conceptos a un formato compatible con la máquina. Además, esos expertos querían disponer de máquinas que pudieran realizar buena parte del proceso de descubrimiento de algoritmos, en lugar de limitarse a ejecutar esos algoritmos. El resultado ha sido un espectro cada vez más amplio de lenguajes de programación que desafía todos los intentos de clasificarlos de manera clara en términos de generaciones.

Paradigmas de programación

La clasificación en generaciones de los lenguajes de programación está basada en una escala lineal (Figura 6.1), en la que la posición de un lenguaje está determinada por el grado en que el usuario de ese lenguaje se ve liberado del mundo de las especifidades técnicas de las computadoras, pudiendo así pensar en términos meramente asociados con el problema que se pretende resolver. En la realidad, el desarrollo de los lenguajes de programación no ha progresado de esta manera, sino de que se ha desarrollado a lo largo de diferentes caminos a medida que han ido apareciendo una serie de enfoques alternativos del proceso

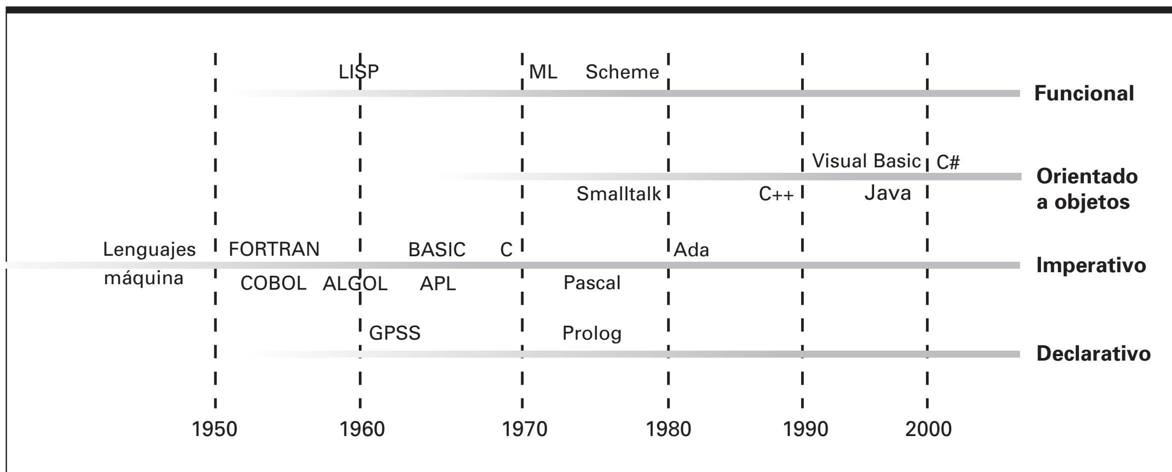
Figura 6.1 Generaciones de lenguajes de programación.



de programación (denominados **paradigmas de programación**). En consecuencia, podemos representar mejor el desarrollo histórico de los lenguajes de programación mediante un diagrama con múltiples rutas paralelas, como se muestra en la Figura 6.2, en la que se muestra cómo las diferentes rutas resultantes de los distintos paradigmas van emergiendo y progresando de manera independiente. En particular, la figura muestra cuatro rutas distintas, que representan los paradigmas funcional, orientado a objetos, imperativo y declarativo, estando colocados los distintos lenguajes asociados con cada uno de los paradigmas de forma tal que la figura indica su nacimiento en relación a otros lenguajes. (Esto no implica que un lenguaje haya evolucionado necesariamente a partir de otro anterior.)

Es preciso observar que aunque los paradigmas identificados en la Figura 6.2 se denominan paradigmas de ***programación***, estas alternativas tienen ramificaciones que van más allá del proceso de programación propiamente dicho. Representan enfoques fundamentalmente distintos para obtener soluciones a los problemas y, por tanto, afectan al proceso completo de desarrollo software. En este sentido, el término ***paradigma de programación*** induce a confusión.Un término más realista sería ***paradigma de desarrollo software.***

Figura 6.2 Evolución de los paradigmas de programación.



El **paradigma imperativo**, también conocido como **paradigma procedimental**, representa el enfoque tradicional del proceso de programación. Es el paradigma en el que está basado nuestro pseudocódigo del Capítulo 5, así como el lenguaje máquina visto en el Capítulo 2. Como su nombre sugiere, el paradigma imperativo define el proceso de programación como el desarrollo de una secuencia de comandos que, al ser ejecutados, manipula los datos para generar el resultado deseado. Por tanto, el paradigma imperativo nos dice que debemos enfocar el proceso de programación determinando un algoritmo para solucionar el problema que nos traemos entre manos y luego expresando dicho algoritmo como una secuencia de sentencias.

A diferencia del paradigma imperativo, el **paradigma declarativo** pide al programador que describa el problema que hay que resolver, en lugar del algoritmo que hay que aplicar. Para ser más precisos, un sistema de programación declarativo aplica un algoritmo preestablecido para resolución de problemas de propósito general con el fin de solucionar los problemas que se le presenten. En un entorno de este tipo, la tarea del programador consiste en desarrollar un enunciado preciso del problema en lugar de describir un algoritmo para la resolución del problema.

Uno de los principales obstáculos a la hora de desarrollar sistemas de programación basados en el paradigma declarativo se encuentra en la necesidad de disponer de un algoritmo subyacente para la resolución del problema. Por esta razón, los primeros lenguajes de programación declarativos tendían a ser de propósito especial por su propia naturaleza, habiendo sido diseñados para su uso en aplicaciones concretas. Por ejemplo, el enfoque declarativo ha sido utilizado durante muchos años para simular sistemas (políticos, económicos, medioambientales, etc.) con el fin de probar hipótesis o de realizar predicciones. En este tipo de entornos, el algoritmo subyacente es básicamente el proceso de simular el paso del tiempo recalculando de forma repetitiva los valores de una serie de parámetros (producto interior bruto, déficit comercial, etc.) a partir de los valores anteriormente calculados. Así, la implementación de un lenguaje declarativo para llevar a cabo simulaciones de este tipo requiere implementar primero un algoritmo que se encargue de llevar a la práctica ese procedimiento repetitivo. Entonces, la única tarea que un programador que use el sistema tiene que llevar a cabo es describir la situación que se desea simular. De esta manera, un meteorólogo no necesita desarrollar un algoritmo para predecir si lloverá o no, sino que simplemente tendrá que describir el estado meteorológico actual, permitiendo al algoritmo de simulación subyacente generar las predicciones del tiempo para los próximos días.

El paradigma declarativo sufrió un tremendo impulso cuando se descubrió que el tema de la lógica formal dentro del campo de las matemáticas proporciona un algoritmo de resolución de problemas simple que es adecuado para su utilización en un sistema de programación declarativa de propósito general. El resultado ha sido que ahora se presta mucha más atención al paradigma declarativo y que ha surgido la denominada **programación lógica**, un tema que veremos en la Sección 6.7.

Otro paradigma de programación es el **paradigma funcional**. En este caso, un programa se ve como una entidad que acepta entradas y genera salidas. Los matemáticos denominan a tales entidades funciones, razón por la que esta técnica recibe el nombre de paradigma funcional. Bajo este paradigma, los

programas se construyen conectando entidades predefinidas más pequeñas (funciones predefinidas), tal que las salidas de cada unidad se utilicen como entradas de otras unidades, de tal forma que al final se obtenga la relación entrada-salida global deseada. En resumen, el proceso de programación bajo el paradigma funcional consiste en construir funciones como conjuntos anidados de otras funciones más simples.

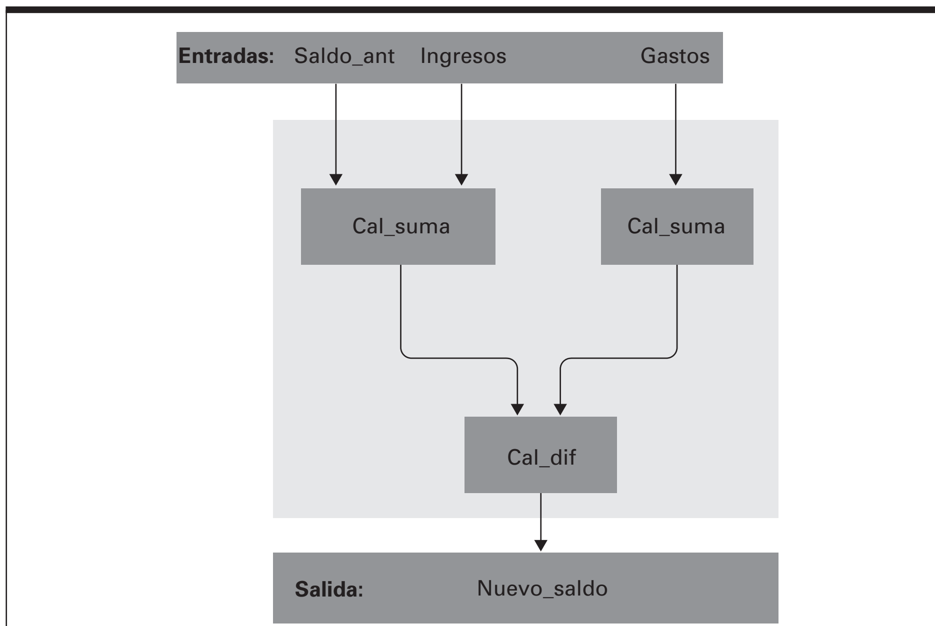
Veamos un ejemplo. La Figura 6.3 muestra cómo se puede construir una función que calcule el saldo de nuestra cuenta corriente a partir de dos funciones más simples. Una de ellas, de nombre `Cal_suma`, acepta valores como entrada y genera la suma de esos valores como salida. La otra, denominada `Cal_dif`, acepta dos valores de entrada y calcula su diferencia. La estructura mostrada en la Figura 6.3 se puede representar en el lenguaje de programación LISP (uno de los principales lenguajes de programación funcionales) mediante la expresión

```
(Cal_dif (Cal_suma Saldo_ant Ingresos) (Cal_suma Gastos))
```

La estructura anidada de esta expresión (como indican los paréntesis) refleja el hecho de que las entradas a la función `Cal_dif` son generadas aplicando `Cal_suma` dos veces. La primera ejecución de `Cal_suma` genera el resultado de sumar todos los `Ingresos` al `Saldo_ant`. La segunda ejecución de `Cal_suma` calcula el total de todas los `Gastos`. Entonces, la función `Cal_dif` utiliza estos resultados para obtener el nuevo saldo de nuestra cuenta bancaria.

Para entender mejor la distinción entre los paradigmas funcional e imperativo, vamos a comparar el programa funcional para calcular el saldo de una cuenta corriente con el siguiente programa de pseudocódigo obtenido ajustándose al paradigma imperativo:

Figura 6.3 Una función para calcular el saldo de una cuenta bancaria construida a partir de funciones más simples.



```
Total_ingresos ← suma de todos los Ingresos
Saldo_temp ← Saldo_ant + Total_ingresos
Total_gastos ← suma de todos los Gastos
Saldo ← Saldo_temp - Total_gastos
```

Observe que este programa imperativo consta de varias sentencias, cada una de las cuales solicita que se realice un cálculo y que el resultado se almacene para un uso posterior. A diferencia de esto, el programa funcional consta de una única sentencia, en la que el resultado de cada cálculo se canaliza de forma inmediata al siguiente. En cierto sentido, el programa imperativo es análogo a un conjunto de fábricas, donde cada fábrica convierte sus materias primas en productos que se guardan en almacenes. Posteriormente, los productos de estos almacenes son suministrados a otras fábricas a medida que los van necesitando. Pero el programa funcional es análogo a un conjunto de fábricas que están coordinadas, de manera que cada una solo fabrica aquellos productos que han sido solicitados por otras fábricas y envía estos productos a sus destinos de forma inmediata sin pasar por un almacenamiento intermedio. Esta eficiencia es una de las ventajas proclamadas por los defensores del paradigma funcional.

Otro paradigma de programación más (y el más importante actualmente en el desarrollo de software) es el **paradigma orientado a objetos**, que está asociado con el proceso de programación denominado **programación orientada a objetos** (OOP, *Object-Oriented Programming*). De acuerdo con este paradigma, un sistema software se ve conceptualmente como un conjunto de unidades, denominadas **objetos**, cada uno de las cuales es capaz de llevar a cabo las acciones que le afectan directamente, así como de solicitar acciones a otros objetos. De forma conjunta, estos objetos interactúan para resolver el problema que tengamos entre manos.

Como ejemplo práctico de la técnica de orientación a objetos, considere la tarea de desarrollar una interfaz gráfica de usuario. En un entorno orientado a objetos, los iconos que aparecen en la pantalla se implementarían como objetos. Cada uno de estos objetos incluiría un conjunto de procedimientos (denominados **métodos** en la jerga de la orientación a objetos) que describirían cómo debe responder el objeto a diversos posibles eventos, como por ejemplo al evento de ser seleccionado por un clic del botón del ratón o al evento de ser arrastrado con el ratón a través de la pantalla. De este modo, el sistema completo se construiría como un conjunto de objetos, cada uno de los cuales sabe cómo responder a los eventos relacionados con él.

Para comparar el paradigma orientado a objetos con el paradigma imperativo, considere un programa en el que se emplee una lista de nombres. En el paradigma imperativo tradicional, esta lista sería simplemente un conjunto de datos. Cualquier unidad de programa que acceda a la lista deberá contener los algoritmos para realizar las manipulaciones requeridas. Sin embargo, en la técnica de orientación a objetos la lista se construiría como un objeto que estaría compuesto por la propia lista y un conjunto de métodos para manipular la lista (esto podría incluir procedimientos para insertar una entrada en la lista, borrar una entrada de la lista, detectar si la lista está vacía y ordenar la lista). Esto quiere decir que cualquier otra unidad de programa que necesitara manipular la lista no contendría ningún algoritmo para realizar las tareas pertinentes. En lugar de ello, haría uso de los procedimientos proporcionados por el objeto. En cierto sentido, en lugar de ordenar la lista, como haríamos en el paradigma

imperativo, lo que una unidad de programa haría en el paradigma de la orientación a objetos sería pedir a la lista que se ordene a sí misma.

Aunque hablaremos con más detalle del paradigma orientado a objetos en la Sección 6.5, su importancia en el panorama actual del desarrollo software nos obliga a incluir el concepto de clase en esta presentación. Con este objetivo, recuerde que un objeto puede estar formado por datos (tal como una lista de nombres) y por un conjunto de métodos para la realización de actividades (como por ejemplo insertar nuevos nombres en la lista). Estas características deben ser descritas mediante sentencias incluidas en el programa que escribamos. Esta descripción de las propiedades de un objeto se denomina **clase**. Una vez construida una clase, puede aplicarse cada vez que haga falta un objeto con dichas características. De este modo, puede haber múltiples objetos basados en la misma clase, es decir, construidos a partir de esa clase. Al igual que sucede con dos gemelos idénticos, estos objetos serían entidades distintas, pero tendrían las mismas características porque han sido construidos a partir de la misma plantilla (la misma clase). (Un objeto que está basado en una clase concreta se dice que es una **instancia** de esa clase.)

La razón de que el paradigma de orientación a objetos haya ganado popularidad es, precisamente, porque los objetos son unidades bien definidas, cuyas descripciones están aisladas en clases reutilizables. De hecho, los defensores de la programación orientada a objetos argumentan que el paradigma de la orientación a objetos proporciona un entorno natural para la técnica de desarrollo software basada en la utilización de "bloques de código reutilizables". Esos expertos tienen la visión de que deberíamos poder disponer de bibliotecas software de clases predefinidas, a partir de las cuales pudieran construirse nuevos sistemas software de la misma forma en que muchos productos tradicionales se construyen a partir de componentes disponibles en el mercado. La construcción y expansión de dicho tipo de bibliotecas es un proceso que está actualmente en marcha, como veremos en el Capítulo 7.

Para terminar, hay que observar que los métodos dentro de un objeto son, en esencia, pequeñas unidades de programa de tipo imperativo. Es decir que la mayoría de los lenguajes de programación basados en el paradigma orientado a objetos contienen muchas de las características que podemos encontrar en los lenguajes imperativos. Por ejemplo, el popular lenguaje orientado a objetos C++ fue desarrollado añadiendo características de orientación a objetos al lenguaje imperativo conocido como C. Además, puesto que Java y C# son derivados de C++, también ellos han heredado ese núcleo de tipo imperativo. En las Secciones 6.2 y 6.3 exploraremos muchas de estas características imperativas y, al hacerlo, hablaremos de una serie de conceptos que son fundamentales para gran parte del software orientado a objetos utilizado hoy día. Después, en la Sección 6.5, consideraremos aquellas características que son exclusivas del paradigma orientado a objetos.

Cuestiones y ejercicios

1. ¿En qué sentido es independiente de la máquina un programa escrito en un lenguaje de tercera generación? ¿En qué sentido continúa siendo dependiente de la máquina?

2. ¿Cuál es la diferencia entre un ensamblador y un compilador?
3. Podemos resumir el paradigma de la programación imperativa diciendo que pone el énfasis en describir un proceso que conduce a la solución del problema que tengamos entre manos. Proporcione un resumen similar de los paradigmas declarativo, funcional y orientado a objetos.
4. ¿En qué sentido están los lenguajes de programación de tercera generación a un nivel mayor que los de generaciones anteriores?

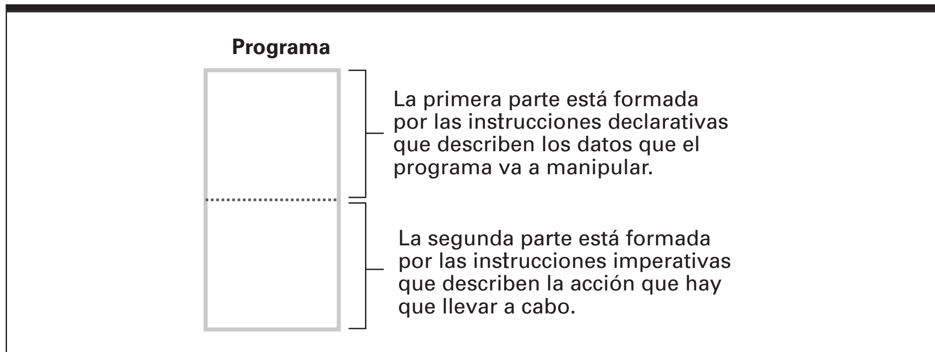
6.2 Conceptos de programación tradicionales

En esta sección vamos a abordar algunos de los conceptos que podemos encontrar tanto en los lenguajes de programación imperativos como en los orientados a objetos. Con este fin, veremos ejemplos de los lenguajes Ada, C, C++, C#, FORTRAN y Java. Nuestro objetivo no es enredarnos en los detalles de ningún lenguaje concreto, sino simplemente ilustrar cómo aparecen características comunes en los lenguajes reales. Por tanto, hemos seleccionado nuestro conjunto de lenguajes para que sea representativo de los que hoy en día existen. C es un lenguaje imperativo de tercera generación. C++ es un lenguaje orientado a objetos que fue desarrollado como extensión del lenguaje C. Java y C# son lenguajes orientados a objetos derivados de C++. (Java fue desarrollado por Sun Microsystems, que más tarde fue adquirida por Oracle, mientras que C# es un producto de Microsoft.) FORTRAN y Ada originalmente fueron desarrollados como lenguajes imperativos de tercera generación, aunque sus versiones más recientes se han ampliado para abarcar buena parte del paradigma de orientación a objetos. En el Apéndice D se proporcionan algunos datos sobre cada uno de estos lenguajes.

Aunque estamos incluyendo lenguajes orientados a objetos como C++, Java y C# entre nuestros lenguajes de ejemplo, abordaremos esta sección como si estuviéramos escribiendo un programa en el paradigma imperativo, porque muchas unidades dentro de un programa orientado a objetos (como por ejemplo los procedimientos que describen cómo debe reaccionar un objeto ante un estímulo externo) son, en esencia, programas cortos de carácter imperativo. Posteriormente, en la Sección 6.5, nos centraremos en las características exclusivas del paradigma de orientación a objetos.

Generalmente, un programa está formado por un conjunto de sentencias que suelen caer dentro de tres categorías distintas: sentencias declarativas, sentencias imperativas y comentarios. Las **sentencias declarativas** describen la terminología personal que se empleará posteriormente en el programa, como por ejemplo los nombres utilizados para hacer referencia a los elementos de datos; las **sentencias imperativas** describen los pasos que componen los algoritmos subyacentes y los **comentarios** mejoran la legibilidad de un programa explicando sus características ocultas en una forma más fácilmente comprensible por parte de las personas. Normalmente, un programa imperativo (o una unidad de programa imperativa dentro de un programa orientado a objetos) puede concebirse como si tuviera la estructura descrita en la Figura 6.4. Comienza con un conjunto de sentencias declarativas que describen los datos que el programa va a manipular. Este material preliminar va seguido por una

Figura 6.4 Composición de un programa (o una unidad de programa) imperativo típico.



serie de sentencias imperativas que describen el algoritmo que hay que ejecutar. Muchos lenguajes permiten ahora que las sentencias declarativas e imperativas se mezclen libremente, pero la distinción conceptual continúa existiendo. Los comentarios están dispersos por el programa según sea necesario clarificarlo.

Teniendo en mente esta estructura, vamos a enfocar nuestro estudio de los conceptos de programación considerando las categorías de sentencias en el orden en el que podemos encontrarlas en un programa, comenzando por los conceptos asociados con las sentencias de declaración.

Variables y tipos de datos

Como se ha mencionado en la Sección 6.1, los lenguajes de programación de alto nivel permiten hacer referencia a las posiciones de la memoria principal mediante nombres descriptivos en lugar de mediante direcciones numéricas. Cada uno de esos nombres se denomina **variable**, en reconocimiento del hecho de que al cambiar el valor almacenado en dicha posición, varía también el valor asociado con el nombre a medida que el programa se ejecuta. Nuestros lenguajes de ejemplo exigen que las variables se identifiquen mediante una sentencia declarativa antes de ser utilizadas en ningún otro lugar del programa. Estas sentencias declarativas también requieren que el programador describa la clase de información que estará almacenada en la posición de memoria asociada con la variable.

Esas clases se conocen con el nombre de **tipo de datos** y definen tanto la manera en que se codifica en memoria el dato como las operaciones que se pueden realizar con esos datos. Por ejemplo, el tipo **integer** hace referencia a datos numéricos compuestos por números enteros, que probablemente se almacenan utilizando notación en complemento a dos. Entre las operaciones que se pueden realizar sobre datos de tipo entero se incluyen las operaciones aritméticas tradicionales y las comparaciones de tamaño relativo, como por ejemplo determinar si uno de los valores es mayor que otro. El tipo **float** (en ocasiones denominado **real**) hace referencia a datos numéricos que pueden contener valores no enteros, probablemente almacenados en notación de punto flotante. Las operaciones que pueden efectuarse sobre los datos de tipo float son similares a las realizadas sobre datos de tipo integer. No obstante, recuerde que la actividad requerida para

Lenguajes de script

Un subconjunto de los lenguajes de programación imperativos son los lenguajes conocidos como **lenguajes de script**. Estos lenguajes se emplean normalmente para llevar a cabo tareas administrativas, más que para desarrollar programas complejos. La expresión de una de esas tareas se conoce con el nombre de **script**, de donde viene el nombre de "lenguaje de script". Por ejemplo, el administrador de un sistema de computadoras podría escribir un script para describir una secuencia de actividades de mantenimiento que haya que llevar a cabo todas las noches. O bien, el usuario de un PC podría escribir un script para controlar la ejecución de una secuencia de programas requeridos para leer imágenes de una cámara digital, indexar las imágenes por fechas y almacenar copias de las mismas en un sistema de almacenamiento permanente. Los lenguajes de script tienen su origen en los lenguajes de control de trabajos de la década de los años 60, lenguajes que se empleaban para controlar al sistema operativo en la planificación de trabajos de procesamiento por lotes (véase la Sección 3.1). Incluso hoy día, muchos consideran que los lenguajes de script son lenguajes para controlar la ejecución de otros programas, lo cual es una visión bastante restrictiva de los lenguajes de script actuales. Entre los ejemplos de los lenguajes de script se incluyen Perl y PHP, ambos muy populares a la hora de controlar aplicaciones web del lado del servidor (véase la Sección 4.3), así como VBScript, que es un dialecto de Visual Basic que fue desarrollado por Microsoft y que se utiliza en entornos Windows.

sumar dos elementos de tipo float es diferente de la necesaria para sumar dos elementos de tipo integer.

Suponga entonces que deseamos utilizar la variable `PesoLimite` en un programa para hacer referencia a un área de la memoria principal que contiene un valor numérico codificado en notación de complemento a dos. En los lenguajes C, C++, Java y C# declararíamos nuestra intención incluyendo cerca del inicio del programa la sentencia

```
int PesoLimite;
```

Esta instrucción significa que "el nombre `PesoLimite` se utilizará posteriormente en el programa para hacer referencia a un área de memoria que contiene un valor almacenado en notación de complemento a dos". Normalmente, pueden declararse en la misma sentencia de declaración varias variables del mismo tipo. Por ejemplo, la sentencia

```
int Altura, Ancho;
```

declararía que tanto `Altura` como `Ancho` son variables de tipo entero (int). Además, la mayoría de los lenguajes permiten asignar un valor inicial a la variable en el momento de declararla. Así

```
int PesoLimite = 100;
```

no solo declara `PesoLimite` como una variable de tipo integer sino que también la asigna el valor inicial 100.

Otros tipos de datos comunes son los datos de caracteres y booleanos. El tipo **character** hace referencia a datos compuestos por un símbolo, que probablemente se almacena utilizando codificación ASCII o Unicode. Entre las ope-

raciones realizadas con dichos datos se incluyen las comparaciones, como por ejemplo determinar si un símbolo aparece antes que otro en orden alfabético, comprobar si una cadena de símbolos aparece dentro de otra y concatenar una cadena de símbolos al final de otra con el objetivo de formar una cadena más larga. La sentencia

```
char Letra, Digito;
```

podría emplearse en los lenguajes C, C++, C# y Java para declarar que las variables `Letra` y `Digito` son de tipo character.

El tipo **Boolean** hace referencia a elementos de datos que solo pueden tomar los valores lógicos verdadero o falso. Entre las operaciones sobre datos de tipo Boolean que se pueden realizar se incluyen las consultas para ver si el valor actual es verdadero o falso. Por ejemplo, si declaráramos la variable `Exceso-Limite` como de tipo Boolean, entonces una sentencia de la forma

```
if (ExcesoLimite) then (...) else (...)
```

sería razonable.

Los tipos de datos que se incluyen como primitivas en un lenguaje de programación, como por ejemplo `int` para los enteros y `char` para los caracteres se denominan **tipos de datos primitivos**. Como hemos visto, los tipos integer, float, character y Boolean son tipos primitivos comunes. Otros tipos de datos que todavía no han llegado a ser tipos primitivos ampliamente aceptados son por ejemplo las imágenes, el audio, el vídeo y el hipertexto. Sin embargo, tipos tales como GIF, JPEG y HTML podrían pronto llegar a ser tan comunes como los enteros o los valores en punto flotante. Posteriormente (Secciones 6.5 y 8.4) veremos cómo el paradigma de orientación a objetos permite a un programador ampliar el repertorio de tipos de datos disponibles, más allá de los tipos primitivos proporcionados por el lenguaje. De hecho, esta capacidad es una de las ventajas más celebradas del paradigma de orientación de objetos.

En resumen, el siguiente fragmento de código, expresado en el lenguaje C y en sus derivados C++, C# y Java, declara las variables `Longitud` y `Ancho` como de tipo float, las variables `Precio`, `Impuesto` y `Total` como de tipo entero y la variable `Simbolo` como de tipo carácter.

```
float  Longitud, Ancho;
int    Precio, Impuesto, Total;
char   Simbolo;
```

En la Sección 6.4 veremos cómo un traductor utiliza el conocimiento extraído de dichas sentencias de declaración como ayuda para traducir un programa de alto nivel a lenguaje máquina. Por el momento vamos a limitarnos a observar que dicha información puede emplearse para identificar errores. Por ejemplo, si un traductor encuentra una sentencia solicitando la suma de dos variables que previamente habían sido declaradas como de tipo Boolean, consideraría probablemente que esa sentencia es un error e informaría de ello al usuario.

Estructuras de datos

Además de con tipos de datos, las variables de un programa suelen asociarse con **estructuras de datos**, que es la forma o disposición conceptual de los

datos. Por ejemplo, el texto se suele entender como una larga cadena de caracteres, mientras que los registros de ventas podrían verse como una tabla rectangular de valores numéricos, en la que cada fila representa las ventas hechas por un determinado empleado y cada columna representa la ventas realizadas en un día determinado.

Una estructura de datos común es la **matriz** (*array*), que es un bloque de elementos del mismo tipo, como por ejemplo una lista unidimensional, una tabla bi-dimensional con filas y columnas o tablas con un número de dimensiones mayor. Para definir una de tales matrices en un programa, muchos lenguajes de programación exigen que la sentencia de declaración en la que se declara el nombre de la matriz también especifique la longitud de cada dimensión de la misma. Por ejemplo, la Figura 6.5 muestra la estructura conceptual declarada por la sentencia

```
int Puntuaciones[2][9];
```

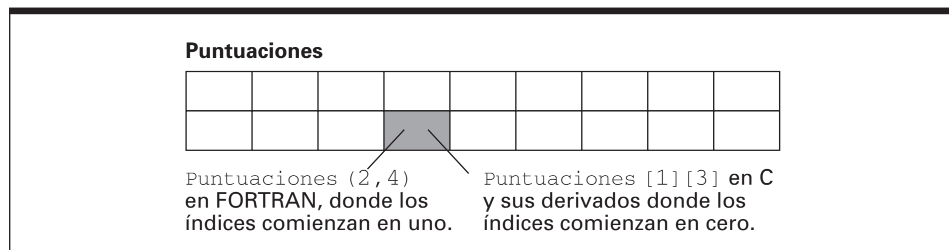
en lenguaje C, que quiere decir "la variable `Puntuaciones` se utilizará en la siguiente unidad de programa para hacer referencia a una matriz bidimensional de enteros compuesta por dos filas y nueve columnas". La misma sentencia en FORTRAN se escribiría como

```
INTEGER Puntuaciones(2,9)
```

Una vez declarada una matriz se puede hacer referencia a ella en cualquier lugar del programa utilizando su nombre. O bien puede identificarse un elemento individual de la matriz por medio de unos valores enteros denominados **índices** que especifican la fila, la columna, etc., deseadas. Sin embargo, el rango de esos índices varía de un lenguaje a otro. Por ejemplo, en C (y sus derivados C++, Java y C#) los índices comienzan en 0, lo que quiere decir que podría hacerse referencia a la entrada situada en la segunda fila y la cuarta columna de la matriz llamada `Puntuaciones` (tal como la hemos declarado anteriormente) mediante `Puntuaciones[1] [3]`, mientras que la entrada situada en la primera fila y la primera columna sería `Puntuaciones[0][0]`. Por el contrario, en un programa FORTRAN los índices comienzan en 1, por lo que para hacer referencia a la entrada situada en la segunda fila y la cuarta columna utilizaríamos `Puntuaciones(2,4)` (véase de nuevo la Figura 6.5).

A diferencia de una matriz en la que todos los elementos de datos son del mismo tipo, un **tipo agregado** (también denominado **estructura** o **registro**) es un bloque de datos en el que los diferentes elementos pueden tener distintos tipos. Por ejemplo, un bloque de datos referido a un empleado podría estar formado por una entrada `Nombre` de tipo character, una entrada `Edad` de tipo entero

Figura 6.5 Una matriz bidimensional con dos filas y nueve columnas.



y una entrada denominada `Categoria` de tipo float. Dicho tipo agregado se declararía en C mediante la sentencia

```
struct  {char   Nombre[25];
         int    Edad;
         float  Categoria;}
        Empleado;
```

que dice que la variable `Empleado` hace referencia a una estructura (utilizaremos la palabra `struct` para referirnos a ella) formada por tres componentes denominados `Nombre` (una cadena de 25 caracteres), `Edad` y `Categoria` (Figura 6.6). Una vez declarado uno de esos agregados, un programador puede utilizar el nombre de la estructura (`Empleado`) para hacer referencia a todo el agregado, o bien puede hacer referencia a **campos** individuales dentro del agregado utilizando el nombre de la estructura, seguido de un punto y del nombre del campo (por ejemplo, `Empleado.Edad`).

En el Capítulo 8 veremos cómo se implementan en la práctica dentro de una computadora estructuras conceptuales tales como la matrices. En particular, veremos que los datos contenidos en una matriz pueden estar dispersos en un área amplia de la memoria principal o del almacenamiento masivo. Esta es la razón por la que decimos que las estructuras de datos son la forma o disposición *conceptual* de los datos. De hecho, la disposición real dentro del sistema de almacenamiento de la computadora puede ser muy diferente de esa disposición conceptual.

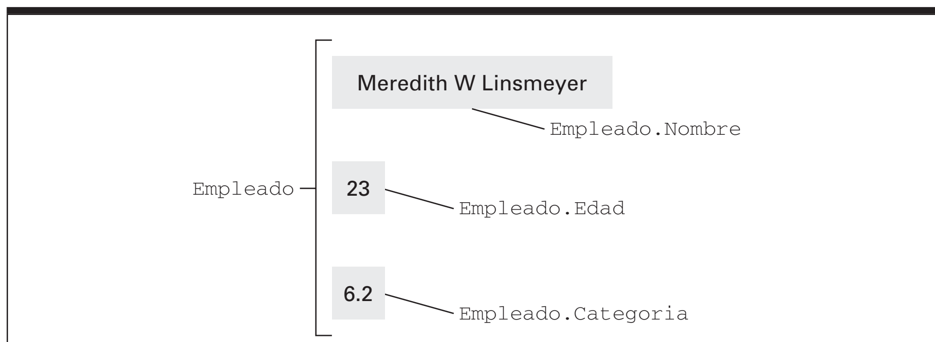
Constantes y literales

En ocasiones es necesario utilizar en un programa un valor fijo predeterminado. Por ejemplo, un programa para controlar el tráfico aéreo en la vecindad de un determinado aeropuerto puede contener numerosas referencias a la altitud de dicho aeropuerto respecto del nivel del mar. Al escribir un programa así, podemos incluir este valor, por ejemplo 645 metros, literalmente cada vez que haga falta. A una aparición explícita de un valor como este, se le denomina **literal**. El uso de literales nos lleva a sentencias de programa tales como

```
AltEfectiva ← Altimetro + 645
```

donde `AltEfectiva` y `Altimetro` se presupone que son variables y `645` es un literal. De acuerdo con esto, esta sentencia pide que se le asigne a la variable

Figura 6.6 Disposición conceptual de la estructura Empleado.



AltEfectiva el resultado de sumar 645 al valor asignado a la variable Altimetro.

En la mayoría de los lenguajes de programación, los literales compuestos por texto se delimitan mediante comillas para distinguirlos de otros componentes del programa. Por ejemplo, la sentencia

```
Apellido ← "Smith"
```

podría utilizarse para asignar el texto "Smith" a la variable Apellido, mientras que la sentencia

```
Apellido ← Smith
```

se emplearía para asignar el valor de la variable Smith a la variable Apellido.

A menudo, el uso de literales no es una buena práctica de programación porque los literales pueden hacer más confuso el significado de las sentencias en las que aparecen. ¿Cómo podría alguien, por ejemplo, leer la sentencia

```
AltEfectiva ← Altimetro + 645
```

y saber lo que representa el valor 645? Además, los literales pueden complicar la tarea de modificar el programa en caso necesario. Si trasladamos nuestro programa de tráfico aéreo a otro aeropuerto, será preciso cambiar todas las referencias a la altitud del aeropuerto. Si se utiliza el literal 645 en cada referencia a dicha altitud, será preciso localizar y modificar cada una de las referencias existentes a lo largo del programa. El problema se complica si el literal 645 aparece también en referencia a algún otro valor distinto de la altitud del aeropuerto. ¿Cómo sabemos qué apariciones del valor 645 hay que modificar y cuáles no?

Para resolver estos problemas, los lenguajes de programación permiten asignar nombres descriptivos a valores específicos no modificables. Cada uno de esos nombres se denomina **constante**. Por ejemplo, en C++ y C#, la sentencia declarativa

```
const int AltAeropuerto = 645;
```

asocia el identificador AltAeropuerto con el valor fijo 645 (que se considera de tipo entero). El concepto similar en Java se expresaría mediante la sentencia

```
final int AltAeropuerto = 645;
```

A continuación de tales declaraciones, podemos utilizar el nombre descriptivo AltAeropuerto en lugar del literal 645. Utilizando esa constante en nuestro pseudocódigo, la sentencia

```
AltEfectiva ← Altimetro + 645
```

podría reescribirse como

```
AltEfectiva ← Altimetro + AltAeropuerto
```

que representa mucho mejor el significado de la sentencia. Además, si se utilizan dichas constantes en lugar de literales y el programa se emplea en otro aeropuerto cuya altitud sea de 267 metros, lo único que hará falta para convertir al nuevo valor todas las referencias a la altitud del aeropuerto será modificar esa única sentencia declarativa en la que hemos definido la constante.

Sentencias de asignación

Una vez declarada la terminología personalizada que se va a utilizar en un programa (como por ejemplo las variables y las constantes), el programador puede empezar a describir los algoritmos necesarios. Esto se hace por medio de sentencias imperativas. La sentencia imperativa más básica es la **sentencia de asignación**, que solicita que se asigne un valor a una variable (o para ser más precisos, que se almacene ese valor en el área de memoria identificada por la variable). Dichas sentencias toman normalmente la forma sintáctica de una variable, seguida por un símbolo que representa la operación de asignación y luego por una expresión que indica el valor que se quiere asignar. La semántica de tales sentencias es que se va a evaluar la expresión y almacenar el resultado como el valor de la variable. Por ejemplo, la sentencia

```
Z = X + Y;
```

en C, C++, C# y Java solicita que se asigne la suma de `X` e `Y` a la variable `Z`. En algunos otros lenguajes (como Ada), la sentencia equivalente tendría el siguiente aspecto

```
Z := X + Y;
```

Observe que estas sentencias solo difieren en la sintaxis del operador de asignación, que en C, C++, C# y Java es simplemente un signo igual, mientras que en Ada es un símbolo de dos puntos seguido de un signo igual. Quizá una mejor notación para el operador de asignación es la que podemos encontrar en APL, un lenguaje diseñado por Kenneth E. Iverson en 1962. (APL significa *A Programming Language*.) En este lenguaje se usa una flecha para representar la asignación. Así, la asignación anterior en APL (así como en nuestro pseudocódigo del Capítulo 5) se expresaría como

```
Z ← X + Y
```

Buena parte de la potencia de las sentencias de asignación proviene del ámbito de las expresiones que pueden aparecer en el lado derecho de la sentencia. En general, puede utilizarse cualquier expresión algebraica, estando los operadores aritméticos de la suma, la resta, la multiplicación y la división representados normalmente por los símbolos +, −, * y /, respectivamente. En algunos lenguajes, se utiliza la combinación ** para representar la operación de exponenciación. Por ejemplo, en Ada la expresión

```
x ** 2
```

representa x^2. Sin embargo, los distintos lenguajes difieren en la forma en la que se interpretan las expresiones algebraicas. Por ejemplo, la expresión 2 * 4 + 6 / 2 podría generar el valor 14 si se la evalúa de derecha a izquierda, o 7 si se evalúa de izquierda a derecha. Estas ambigüedades se suelen resolver mediante una serie de reglas de **precedencia de los operadores**, lo que quiere decir que ciertas operaciones tienen precedencia sobre otras. Las reglas tradicionales del álgebra dictan que la multiplicación y la división tienen precedencia sobre la suma y la resta. Es decir, las multiplicaciones y las divisiones se efectúan antes que las sumas y las restas. De acuerdo con este convenio, la expresión anterior generaría el valor 11. En la mayoría de los lenguajes pueden utilizarse paréntesis para saltarse la precedencia de operadores del lenguaje. Por ejemplo, 2 * (4 + 6) / 2 daría como resultado el valor 10.

Muchos lenguajes de programación permiten utilizar un mismo símbolo para representar más de una operación. En estos casos, el significado del símbolo queda determinado por el tipo de datos de los operandos. Por ejemplo, el símbolo + indica tradicionalmente la suma cuando sus operandos son numéricos, pero en algunos lenguajes, como por ejemplo en Java, el mismo símbolo indica la concatenación cuando sus operandos son cadenas de caracteres. Es decir, el resultado de la expresión

```
"abra" + "cadabra"
```

es *abracadabra*. Este tipo de usos múltiples de un mismo símbolo de operación se denomina **sobrecarga**. Mientras que muchos lenguajes proporcionan un mecanismo integrado de sobrecarga para algunos operadores comunes, otros lenguajes como Ada, C++ y C# permiten a los programadores definir significados sobrecargados adicionales o incluso añadir operadores nuevos.

Sentencia de control

Una **sentencia de control** modifica la secuencia de ejecución del programa. De todas las estructuras de programación, las que pertenecen a este grupo son las que han recibido probablemente la mayor atención y las que han generado mayor controversia. El villano principal de la historia es la sentencia de control más simple de todas, la sentencia `goto` (sentencia de salto). Esta sentencia proporciona un medio de dirigir la secuencia de ejecución a otra ubicación que haya sido etiquetada con este propósito mediante un nombre o un número. Por tanto, no es nada más que una aplicación directa de la instrucción JUMP de nivel máquina. El problema con esta funcionalidad en un lenguaje de programación de alto nivel es que permite a los programadores escribir estructuras enormemente confusas como

```
     goto 40
20   Aplicar procedimiento Escapar
     goto 70
40   if (NivelKriptonita < DosisLetal) then goto 60
     goto 20
60   Aplicar procedimiento RescatarDamisela
70   ...
```

cuando una única sentencia como

```
if (NivelKriptonita < DosisLetal)
     then (aplicar procedimiento RescatarDamisela)
     else (aplicar procedimiento Escapar)
```

hubiera valido perfectamente.

Para evitar este tipo de estructuras complejas, los lenguajes modernos se diseñan con sentencias de control que permiten expresar un patrón de bifurcación completo mediante una única estructura léxica. La elección de qué sentencias de control incorporar en un lenguaje es una decisión de diseño. El objetivo es proporcionar un lenguaje que no solo permita expresar los algoritmos en una forma legible, sino que también ayude al programador a conseguir esa legibilidad. Esto se hace restringiendo el uso de aquellas características que han llevado históricamente al desarrollo de programas confusos, al mismo

Culturas de los lenguajes de programación

Al igual que sucede con los lenguajes naturales, los usuarios de los distintos lenguajes de programación tienden a desarrollar diferencias culturales y suelen enzarzarse en debates sobre las respectivas ventajas de los distintos enfoques. En ocasiones, estas diferencias son tan significativas como, por ejemplo, cuando se discute acerca de diferentes paradigmas de programación. En otros casos, las diferencias son mucho más sutiles. Por ejemplo, mientras que en este libro distinguimos entre procedimientos y funciones (Sección 6.3), los programadores en C utilizan el término función para designar ambos conceptos. Esto se debe a que en un programa C, un procedimiento se considera como una función que no devuelve ningún valor. Otro ejemplo similar sería que los programadores de C++ se refieren a un procedimiento dentro de un objeto denominándolo función miembro, mientras que el término genérico para este concepto sería método. Esta discrepancia tiene su origen en el hecho de que C++ fue desarrollado como extensión de C. Otra diferencia cultural es que los programas en Ada suelen escribirse poniendo las palabras reservadas en mayúsuculas o en negrita, una tradición que no suelen adoptar los usuarios de C, C++, C#, FORTRAN o Java.

Aunque este libro es neutral en lo que al lenguaje de programación respecta y utiliza terminología genérica, cada ejemplo específico se presenta de forma tal que sea compatible con el estilo del lenguaje implicado. A medida que se encuentre con esos ejemplos, debe recordar que los estamos presentando como ejemplos de la forma en que las ideas genéricas aparecen en los lenguajes reales, no como medio de enseñar los detalles de ningún lenguaje concreto. Trate de fijarse en el bosque en lugar de fijar su atención en los árboles.

tiempo que se anima a utilizar otras características mejor diseñadas. El resultado es una práctica que se conoce con el nombre de **programación estructurada**, que abarca una metodología de diseño organizada en combinación con el uso apropiado de las sentencias de control del lenguaje. La idea es generar un programa que pueda entenderse fácilmente y que también pueda comprobarse fácilmente para ver si cumple con sus especificaciones.

Ya nos hemos encontrado con dos estructuras de bifurcación populares en nuestro pseudocódigo del Capítulo 5, representadas por las sentencias `if-then-else` y `while`. Estas estructuras de bifurcación están presentes en casi todos los lenguajes imperativos, funcionales u orientados a objetos. Para ser más precisos, las sentencias de pseudocódigo

```
if (condición)
  then (sentenciaA)
  else (sentenciaB)
```

y

```
while (condición) do
   (cuerpo del bucle)
```

se escribirían como

```
if (condición) sentenciaA
  else sentenciaB;
```

y

```
while (condición)
   {cuerpo del bucle}
```

en C, C++, C# y Java. Observe que el hecho de que estas sentencias sean idénticas en los cuatro lenguajes es consecuencia de que C++, C# y Java son extensiones orientadas a objetos del lenguaje imperativo C. Por el contrario, las sentencias correspondientes se escribirían como

```
IF condición THEN
  sentenciaA;
ELSE
  sentenciaB;
END IF
```

y

```
WHILE condición LOOP
  cuerpo del bucle
END LOOP;
```

en el lenguaje Ada.

Otra estructura de bifurcación común suele representarse mediante una sentencia `switch` o `case`. Proporciona un medio de seleccionar una secuencia de sentencias entre varias opciones posibles, dependiendo del valor asignado a una variable concreta. Por ejemplo, la sentencia

```
switch (variable) {
  case 'A': sentenciaA; break;
  case 'B': sentenciaB; break;
  case 'C': sentenciaC; break;
  default: sentenciaD}
```

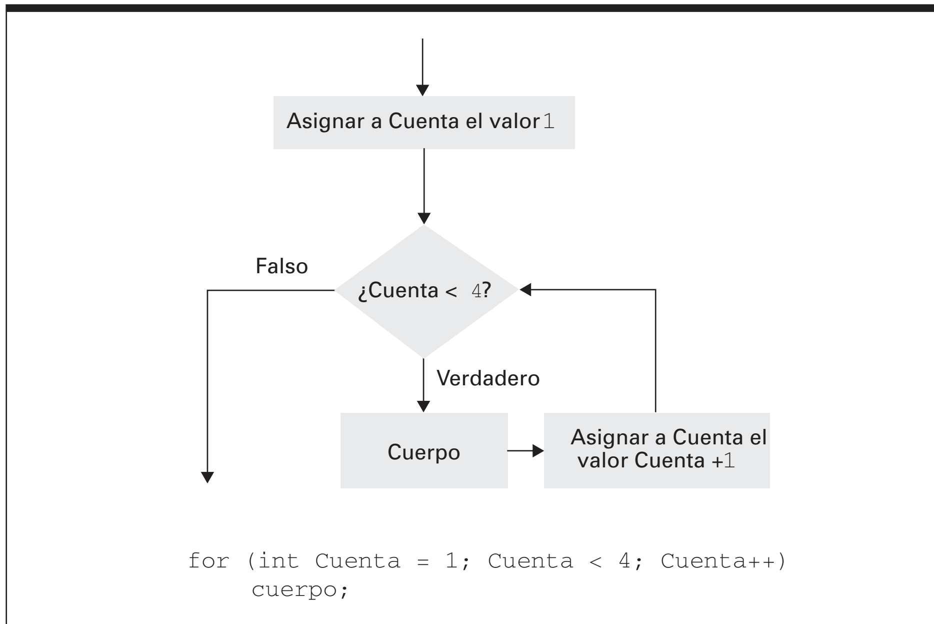
en C, C++, C# y Java solicita que se ejecute la *sentenciaA*, la *sentenciaB*, o la *sentenciaC* dependiendo de si el valor actual de la *variable* es A, B o C, respectivamente, o la ejecución de la *sentenciaD* si el valor de la *variable* es cualquier otro. La misma estructura en Ada se expresaría como

```
CASE variable IS
  WHEN 'A'=> sentenciaA;
  WHEN 'B'=> sentenciaB;
  WHEN 'C'=> sentenciaC;
  WHEN OTHERS=> sentenciaD;
END CASE
```

Otra estructura de control común adicional, a menudo denominada estructura ***for*** es la mostrada en la Figura 6.7 junto con su representación en C++, C# y Java. Se trata de una estructura iterativa similar a la de la sentencia `while` de nuestro pseudocódigo. La diferencia es que todas las tareas de inicialización, actualización y comprobación del bucle se incorporan en una única sentencia. Esta sentencia es cómoda cuando el cuerpo del bucle tiene que ejecutarse una vez por cada uno de los valores contenidos en un rango específico. En particular, las sentencias de la Figura 6.7 indican que el cuerpo del bucle debe ejecutarse repetidamente, primero con un valor de `Cuenta` igual a 1, luego con un valor de `Cuenta` igual a 2 y finalmente con un valor de `Cuenta` igual a 3.

La lección que hay que extraer de los ejemplos que hemos citado es que en toda la gama de lenguajes de programación imperativos y orientados a objetos aparece una serie de estructuras de bifurcación comunes, con ligeras variaciones. Un resultado hasta cierto punto sorprendente de las investigaciones en el

Figura 6.7 La estructura del bucle for y su representación en C++, C# y Java.




```
for (int Cuenta = 1; Cuenta < 4; Cuenta++)
    cuerpo;
```

campo de la Ciencias de la computación es que solo son necesarias unas cuantas de estas estructuras para garantizar que un lenguaje de programación proporcione un medio de expresar una solución a cualquier problema que disponga de solución algorítmica. Investigaremos esta afirmación en el Capítulo 12. Por ahora, resaltemos simplemente que aprender un lenguaje de programación no es una tarea interminable de aprendizaje de distintas sentencias de control. La mayoría de las estructuras de control que podemos encontrar en los lenguajes de programación actuales son, en esencia, variaciones de las que hemos presentado aquí.

Comentarios

Independientemente de lo bien que esté diseñado un lenguaje de programación y de lo bien que se apliquen las características del lenguaje en un programa, suele ser útil (y en ocasiones obligatorio) proporcionar información adicional, para cuando una persona trate de leer y entender el programa. Por esta razón, los lenguajes de programación proporcionan formas de insertar enunciados explicativos, denominados **comentarios**, dentro de un programa. Estas sentencias son ignoradas por los traductores y, por tanto, su presencia o ausencia no afecta al programa desde el punto de vista de la máquina. La versión en lenguaje máquina del programa generado por un traductor será la misma con y sin comentarios, pero la información proporcionada por esas sentencias de comentario constituye una importante parte del programa, desde la perspectiva de los seres humanos. Sin esa documentación, los programas largos y complejos podrían escapar a la comprensión de un programador.

Existen dos formas habituales de insertar comentarios en un programa. Una consiste en rodear todo el comentario mediante marcadores especiales,

uno al principio del comentario y otro al final del mismo. La otra forma consiste en marcar únicamente el principio del comentario y permitir que este ocupe el resto la línea, a la derecha del marcador. Podemos encontrar ejemplos de ambas técnicas en C++, C# y Java. Estos lenguajes permiten encerrar los comentarios entre /* y */, pero también permiten que un comentario comience con // y se extienda a lo largo del resto de la línea. Así, tanto

```
/* Esto es un comentario. */
```

como

```
// Esto es un comentario.
```

son sentencias de comentario válidas.

Es necesario decir algunas cosas acerca de qué es lo que constituye un comentario con significado. Los programadores novatos, cuando se les dice que incluyan comentarios con propósitos de documentación interna, tienden a poner a continuación de una sentencia de programa tal como

```
AnguloAproximacion = AnguloDeslizamiento + InclinacionHiperEspacio;
```

un comentario tal como "Calcula AnguloAproximación sumando InclinacionHiperEspacio y AnguloDeslizamiento". Esa redundancia añade longitud, más que claridad, a un programa. El propósito de un comentario es explicar el programa, no repetirlo. Un comentario más apropiado en este caso sería explicar por qué estamos calculando AnguloAproximación (en caso de que no sea obvio). Por ejemplo, el comentario "AnguloAproximación se utiliza posteriormente para calcular VelocidadCampoFuerza y no se necesita después", sería más útil que el comentario anterior.

Además, los comentarios dispersos entre las sentencias de un programa pueden en ocasiones dificultar nuestra capacidad de seguir el flujo del programa y hacen así, por tanto, más difícil de comprender el programa que si no se hubieran incluido comentarios. Una buena técnica consiste en agrupar los comentarios relacionados con una misma unidad de programa en un único lugar, quizá al principio de dicha unidad. Esto proporciona un lugar centralizado en el que el lector de la unidad de programa puede buscar las explicaciones pertinentes. También proporciona una ubicación en la que describir el propósito y las características generales de la unidad de programa. Si se adopta este formato para todas las unidades de programa, se proporciona a todo el programa un grado de uniformidad en el que cada unidad estará compuesta de un bloque de sentencias explicativas, seguido por la presentación formal de la unidad de programa. Tal uniformidad en un programa mejora la legibilidad.

Cuestiones y ejercicios

1. ¿Por qué se considera un mejor estilo de programación el uso de una constante, en lugar de un literal?
2. ¿Cuál es la diferencia entre una sentencia declarativa y una sentencia imperativa?
3. Enumere algunos tipos de datos comunes.

4. Identifique algunas estructuras de control comunes que podemos encontrar en los lenguajes de programación imperativos y orientados a objetos.
5. ¿Cuál es la diferencia entre un array y un tipo agregado?

6.3 Procedimientos

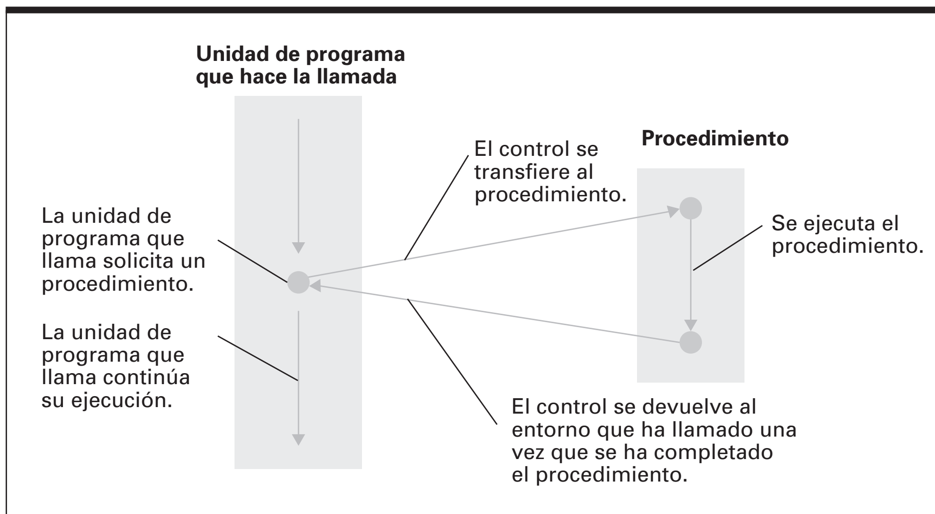
En los capítulos anteriores hemos visto las ventajas de dividir los programas de gran tamaño en unidades más manejables. En esta sección, vamos a centrarnos en el concepto de procedimiento, que es la técnica principal para obtener una representación modular de un programa en un lenguaje imperativo. Además, en los lenguajes orientados a objetos, los programadores, por medio de los procedimientos, pueden especificar cómo deben responder los objetos a los diversos estímulos.

Procedimientos

Un **procedimiento**, en sentido genérico, es un conjunto de sentencias para realizar una tarea que puede ser utilizado como herramienta abstracta por otras unidades de programa. El control se transfiere al procedimiento en el mismo momento en que se solicitan sus servicios y luego se devuelve ese control a la unidad de programa original después de que el procedimiento haya finalizado (Figura 6.8). El proceso de trasferir el control a un procedimiento se denomina a menudo ***llamada al*** o ***invocación del*** procedimiento. Denominaremos a la unidad de programa que solicita la ejecución de un procedimiento unidad que realiza la llamada o unidad llamante.

Como en nuestro pseudocódigo del Capítulo 5, los procedimientos normalmente pueden escribirse como unidades de programa individuales. La unidad comienza como una sentencia, conocida como **cabecera del procedimiento**,

Figura 6.8 El flujo de control relativo a un procedimiento.



que identifica, entre otras cosas, el nombre del procedimiento. A continuación de esta cabecera se incluyen las sentencias que definen los detalles del procedimiento. Estas sentencias suelen estar organizadas de la misma forma que un programa imperativo tradicional, comenzando con las instrucciones de declaración que describen las variables utilizadas en el procedimiento e incluyendo luego las sentencias imperativas que describen los pasos que hay que realizar cuando el procedimiento se ejecute.

Como regla general, una variable declarada dentro de un procedimiento es una **variable local**, lo que quiere decir que solo se puede hacer referencia a ella dentro de dicho procedimiento. Esto elimina cualquier confusión que pueda aparecer si dos procedimientos escritos de forma independiente utilizan por casualidad variables con el mismo nombre. (La parte de un programa en la que se puede hacer referencia a una variable se denomina **ámbito** de la variable. Por tanto, el ámbito de una variable local es el procedimiento en el que se la declara. Las variables cuyo ámbito no está restringido a una parte concreta de un programa son **variables globales**. La mayoría de los lenguajes de programación proporcionan un medio de especificar si una variable es local o global.)

A diferencia de nuestro pseudocódigo del Capítulo 5, en el que solicitábamos la ejecución de un procedimiento mediante una sentencia tal como "`Aplicar el procedimiento DeactivarKriptonita`", la mayoría de los lenguajes modernos de programación, como ya hemos dicho, permiten invocar procedimientos simplemente especificando su nombre. Por ejemplo, si `ObtenerNombres`, `OrdenarNombres` y `EscribirNombres` fueran los nombres de procedimientos para la adquisición, ordenación e impresión de una lista de nombres, entonces un programa que obtuviera, ordenara e imprimiera la lista podría escribirse como

```
ObtenerNombres;
OrdenarNombres;
EscribirNombres;
```

en lugar de

```
Aplicar el procedimiento ObtenerNombres.
Aplicar el procedimiento OrdenarNombres.
Aplicar el procedimiento EscribirNombres.
```

Observe que al asignar a cada procedimiento un nombre que indica la acción que lleva a cabo, esta forma condensada aparece como una secuencia de comandos que refleja perfectamente el significado del programa.

Parámetros

Los procedimientos suelen escribirse utilizando términos genéricos, que solo se hacen específicos cuando se aplica el procedimiento. Por ejemplo, la Figura 5.11 del capítulo anterior está expresada en términos de una lista genérica, en lugar de en términos de una lista específica. En nuestro pseudocódigo, hemos decidido identificar dichos términos genéricos dentro de paréntesis en la cabecera del procedimiento. Así, el procedimiento de la Figura 5.11 comienza con la cabecera

```
procedure Ordenar (Lista)
```

y luego continúa describiendo el proceso de ordenación, utilizando el término `Lista` para hacer referencia a la lista que estamos ordenando. Si deseáramos aplicar el procedimiento a la ordenación de una lista de invitaciones de boda, simplemente necesitaríamos seguir las instrucciones indicadas en el procedimiento, asumiendo que el término genérico `Lista` hace referencia a lista de invitados de boda. Sin embargo, si quisiéramos ordenar la lista de miembros de una asociación, simplemente tendríamos que interpretar que el término genérico `Lista` se está refiriendo a esa lista de miembros.

Estos términos genéricos de los procedimientos se denominan **parámetros**. Para ser más precisos, los términos utilizados dentro del procedimiento se denominan **parámetros formales** y el valor concreto que adquieren esos parámetros formales en el momento de aplicar el procedimiento se denominan **parámetros reales** o argumentos. En cierto sentido, los parámetros formales representan como una especie de ranuras dentro del procedimiento en las que se insertan los parámetros reales en el momento de solicitar la ejecución de ese procedimiento.

Como en el caso de nuestro pseudocódigo, la mayoría de los lenguajes de programación requieren que en el momento de definir un procedimiento, se enumeren entre paréntesis en la cabecera del procedimiento los parámetros formales. Por ejemplo, la Figura 6.9 presenta la definición de un procedimiento llamado `PoblacionEstimada` tal como podría escribirse en el lenguaje de programación C. El procedimiento espera que se le proporcione una tasa de crecimiento anual específica en el momento de invocarlo. Basándose en esta tasa, el procedimiento calcula la población estimada de una especie durante los siguien-

Figura 6.9 El procedimiento PoblacionEstimada escrito en el lenguaje de programación C.

Iniciar la cabecera con el término "void" es la forma en la que un programador de C especifica que la unidad de programa es un procedimiento en lugar de una función. Estudiaremos las funciones enseguida.

La lista de parámetros formales. Observe que C, como muchos otros lenguajes de programación, requiere que se especifique el tipo de dato de cada parámetro.

```
void  PoblacionEstimada  (float TasaCrec)

{ int Year;

  Poblacion[0] = 100.0;
  for (Year = 0; Year =< 10; Year++)
  Poblacion[Year+1] = Poblacion[Year] + (Poblacion[Year] * TasaCrec);
}
```

Esta sentencia declara una variable local llamada Year.

Estas sentencias describen cómo se calculan las poblaciones y se almacenan en un array global denominado Poblacion.

tes 10 años, suponiendo una población inicial igual a 100 individuos y almacena estos valores en una matriz global de nombre `Poblacion`.

La mayoría de los lenguajes de programación utilizan también notación de paréntesis para identificar los parámetros reales en el momento de invocar un procedimiento. Es decir, la sentencia en la que se solicita la ejecución de un procedimiento está compuesta por el nombre del procedimiento seguido de una lista de los parámetros reales encerrada entre paréntesis. Por tanto, en lugar de una sentencia como

```
Aplicar PoblacionEstimada usando una tasa de crecimiento de 0.03
```

que utilizaríamos en nuestro pseudocódigo, en un programa en C usaríamos la sentencia

```
PoblacionEstimada(0.03);
```

para llamar al procedimiento `PoblacionEstimada` de la Figura 6.9 utilizando una tasa de crecimiento igual a 0.03.

Cuando interviene más de un parámetro, los parámetros reales se asocian, uno a uno, con los parámetros formales enumerados en la cabecera del procedimiento: el primer parámetro real se asocia con el primer parámetro formal, y así sucesivamente. Entonces, los valores de los parámetros reales son transferidos efectivamente a sus correspondientes parámetros formales, después de lo cual se ejecuta el procedimiento.

Para dejar claro este punto, suponga que definimos el procedimiento `ImprimirCheque` con una cabecera como

```
procedure ImprimirCheque(Beneficiario, Importe)
```

donde `Beneficiario` e `Importe` son parámetros formales utilizados dentro del procedimiento para hacer referencia a la persona a la que hay que pagar un cheque y a la cantidad de dinero que hay que abonar, respectivamente. Entonces, si invocamos al procedimiento mediante la sentencia

```
ImprimirCheque("Juan Diez", 150)
```

dicho procedimiento se ejecutará asociando el parámetro formal `Beneficiario` con el parámetro real Juan Diez y el parámetro formal `Importe` con el valor 150. Sin embargo, si invocáramos al procedimiento con la sentencia

```
ImprimirCheque(150, "Juan Diez")
```

haría que el valor 150 fuera asignado al parámetro formal `Beneficiario` y el nombre Juan Diez fuera asignado al parámetro formal `Importe`, lo que nos llevaría a obtener un resultado erróneo.

Los distintos lenguajes de programación llevan a cabo la tarea de transferir datos entre los parámetros reales y formales de diversas formas. En algunos lenguajes, se genera un duplicado de los datos representados por los parámetros reales y ese duplicado se entrega al procedimiento. Con esta técnica, cualquier alteración que realice el procedimiento sobre los datos de entrada solo se reflejará en el duplicado; los datos existentes en la unidad de programa que hace la llamada nunca se verán modificados. En ocasiones decimos que tales parámetros se **pasan por valor**. Observe que el pasar parámetros por valor protege a los datos contenidos en la unidad llamante, ya que resulta imposible que sean modificados por un error en un procedimiento mal diseñado. Por

ejemplo, si la unidad llamante pasara el nombre de un empleado a un procedimiento, evidentemente no deseará que ese procedimiento modifique dicho nombre.

Lamentablemente, pasar parámetros por valor es ineficiente cuando esos parámetros representan grandes bloques de datos. Otra técnica más eficiente consiste en conceder al procedimiento un acceso directo a los parámetros reales, indicándole las direcciones de esos parámetros reales dentro de la unidad de programa llamante. En este caso, decimos que los parámetros se **pasan por referencia**. Observe que pasar los parámetros por referencia permite al procedimiento modificar los datos que residen en el entorno que llama. Esta técnica sería deseable en el caso de un procedimiento que ordenara una lista, ya que precisamente el invocar ese tipo de procedimiento es para realizar modificaciones en la lista.

Por ejemplo, supongamos que se define el procedimiento `Demo` como

```
procedure Demo (Formal)
Formal ← Formal + 1;
```

Suponga además que asignamos el valor 5 a la variable `Real` y que invocamos al procedimiento `Demo` con la sentencia

```
Demo(Real)
```

Entonces, si pasáramos los parámetros por valor, el cambio efectuado en la variable `Formal` dentro del procedimiento no se vería reflejado en la variable `Real` (Figura 6.10). Pero si los parámetros se pasan por referencia, el valor de `Real` se vería incrementado en una unidad (Figura 6.11).

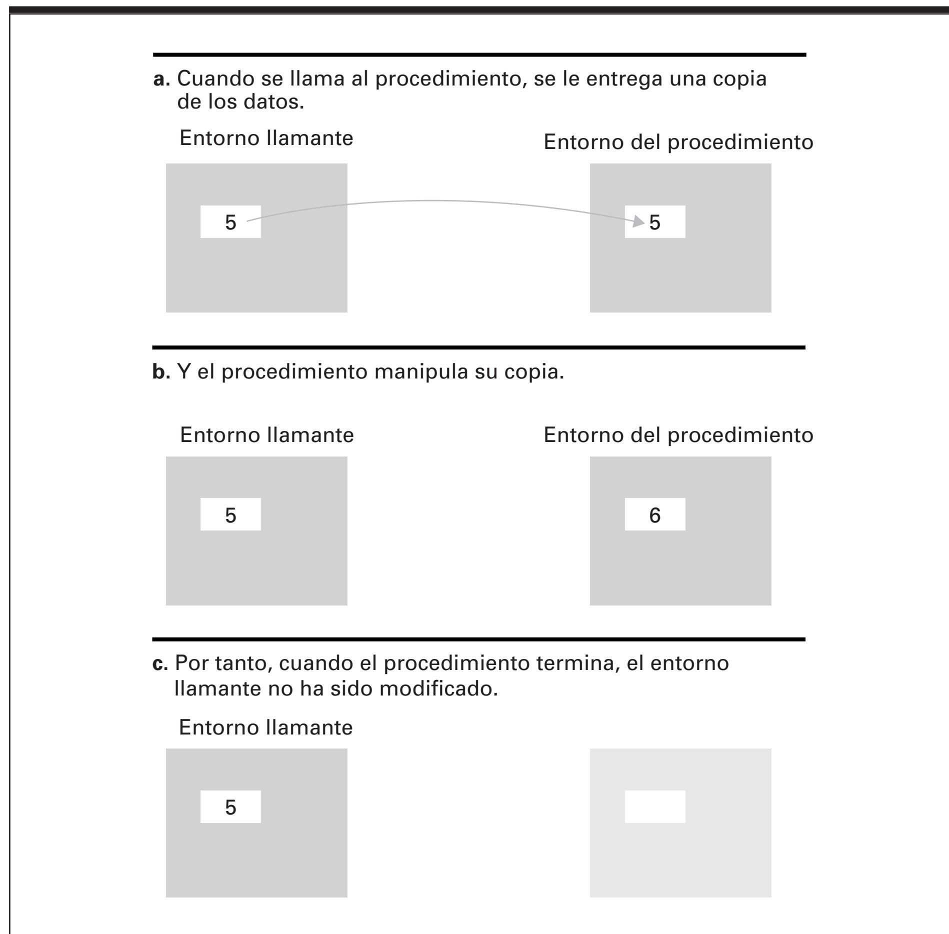
Los distintos lenguajes de programación proporcionan diferentes técnicas para pasar los parámetros, pero en todos los casos el uso de parámetros permite que un procedimiento sea escrito en sentido genérico y aplicado a datos específicos en el momento apropiado.

Visual Basic

Visual Basic es un lenguaje de programación orientado a objetos que fue desarrollado por Microsoft como una herramienta mediante la que los usuarios del sistema operativo Microsoft Windows podían desarrollar sus propias aplicaciones GUI. Realmente, Visual Basic es más que un lenguaje, es un paquete de desarrollo software completo que permite a un programador construir aplicaciones a partir de componentes predefinidos (como por ejemplo botones, casillas de verificación, cuadros de texto, barras de desplazamiento, etc.) y personalizar estos componentes describiendo cómo deben actuar ante diversos sucesos. Por ejemplo, en el caso de un botón, el programador describirá lo que tiene que suceder cuando se haga clic en ese botón. En el Capítulo 7 veremos que esta estrategia de construcción de software a partir de componentes predefinidos representa la tendencia actual dentro del campo de las técnicas de desarrollo software.

La popularidad del sistema operativo Windows, combinada con la comodidad del paquete de desarrollo Visual Basic, ha hecho que Visual Basic se convierta en un lenguaje de programación ampliamente utilizado.

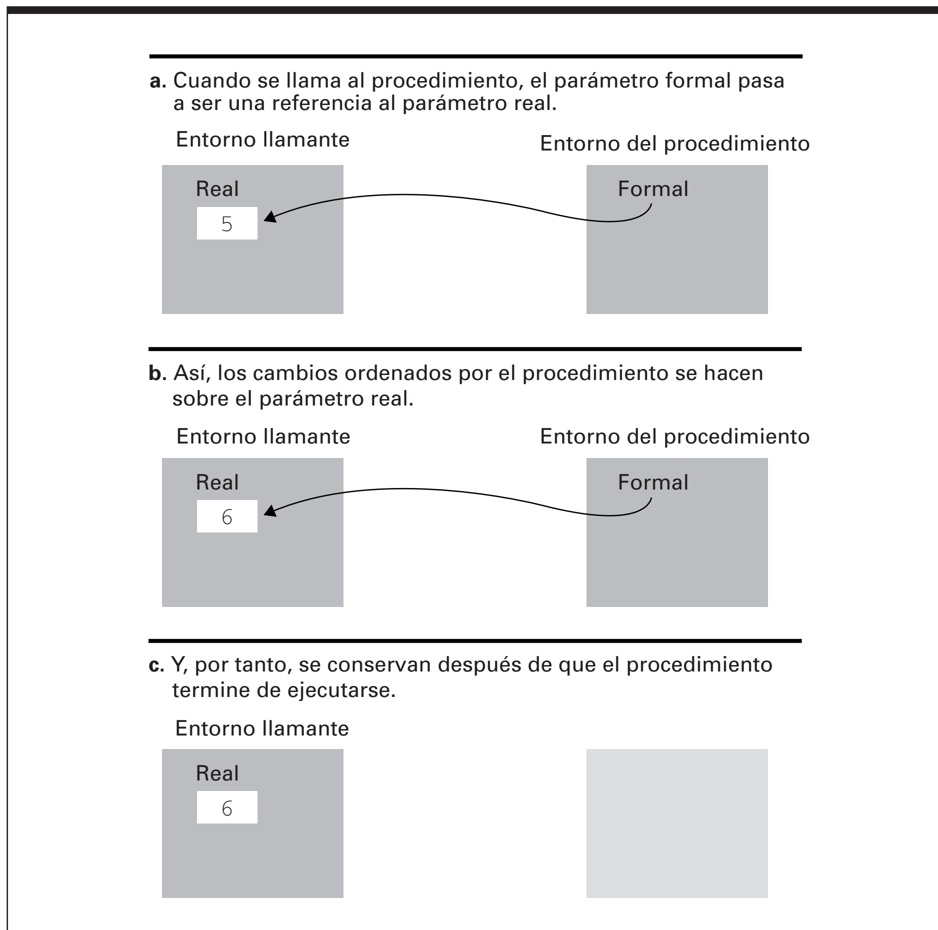
Figura 6.10 Ejecución del procedimiento Demo y paso de parámetros por valor.



Funciones

Hagamos un alto en el camino para dirigir nuestra atención hacia una ligera variante del concepto de procedimiento, que puede encontrarse en muchos lenguajes de programación. En ocasiones, el propósito de un procedimiento es generar un valor, más que llevar a cabo una acción. (Considere la sutil diferencia entre un procedimiento cuyo propósito sea estimar el número de productos que van a venderse y otro procedimiento utilizado para jugar una partida de algún juego; el énfasis en el primero de los dos está en generar un valor, mientras que en el segundo está en realizar una acción.) Si el propósito es generar un valor, el "procedimiento" puede implementarse como una función. En este caso, el término **función** hace referencia a una unidad de programa similar a un procedimiento, excepto porque devuelve un valor a la unidad de programa que le ha llamado como "valor de la función". Es decir, como consecuencia de ejecutar la función, se calculará un valor y se devolverá a la unidad de programa llamante. Este valor podrá ser entonces almacenado en una variable para una referencia posterior o bien puede utilizarse de

Figura 6.11 Ejecución del procedimiento Demo y paso de parámetros por referencia.



manera inmediata en un cálculo. Por ejemplo, un programador de C, C++, Java o C# podría escribir

```
VentasEstimadasEnero = EstimacionVentas(Enero);
```

para solicitar que el resultado de aplicar la función `EstimacionVentas` sea asignado a la variable `VentasEstimadasEnero`, con el fin de determinar cuántos productos se proyectan vender en el mes de enero. O bien, el programador podría escribir

```
if (VentasEstimadasEnero < EstimacionVentas(Enero)) ...
    else ...
```

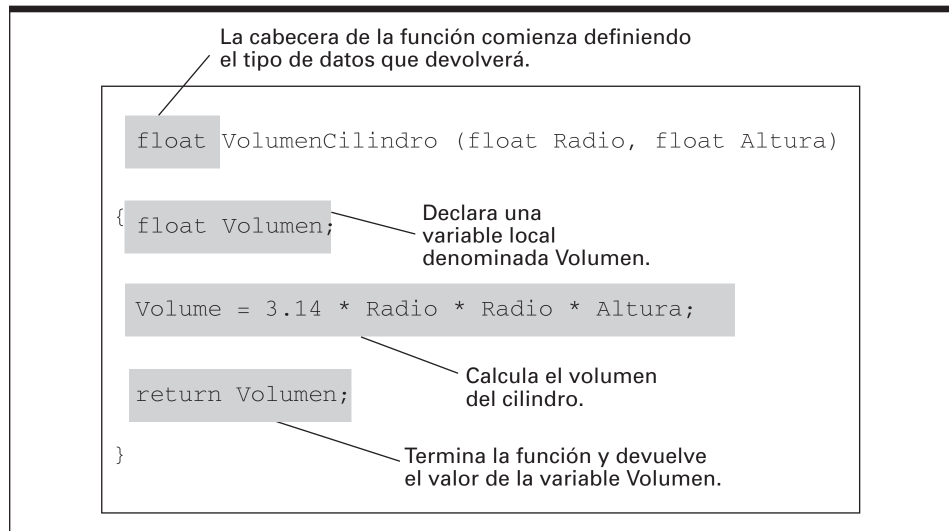
para llevar a cabo diferentes acciones dependiendo de si se espera que las ventas de este mes de enero sean mayores que las del mes de enero anterior. Observe que en el segundo caso, el valor calculado por la función se utiliza para determinar qué rama hay que ejecutar, pero nunca se almacena.

Las funciones se definen dentro de un programa de forma muy similar a como se definen los procedimientos. La diferencia es que la cabecera de una

Sistemas software dirigidos por eventos

En el texto hemos considerado una serie de casos en los que se activan procedimientos como resultado de la ejecución de sentencias en alguna otra parte del programa que llaman explícitamente a esos procedimientos. Sin embargo, hay casos en los que hay procedimientos que se activan de forma implícita debido a que se ha producido un determinado evento. Podemos encontrar ejemplos de esto en las interfaces GUI, en las que el procedimiento que describe lo que debe suceder cuando se hace clic sobre un botón no se activa por una llamada desde otra unidad de programa, sino como resultado de haber hecho clic sobre el botón. Los sistemas software en los que los procedimientos se activan mediante eventos, en lugar de mediante solicitudes explícitas, se conocen como sistemas **dirigidos por eventos**. En resumen, un sistema software dirigido por eventos consiste en procedimientos que describen lo que debe ocurrir como resultado de distintos eventos. Cuando se ejecuta el sistema, estos procedimientos permanecen en estado durmiente hasta que se producen sus respectivos eventos, momento en el que se activan, realizan su tarea y vuelven al estado de durmiente.

Figura 6.12 La función VolumenCilindro escrita en el lenguaje de programación C.



función normalmente comienza especificando el tipo de dato del valor que hay que devolver; además, la definición de la función suele terminar con una sentencia de retorno (`return`) en la que se especifica el valor que hay que devolver. La Figura 6.12 muestra la definición de una función denominada `VolumenCilindro` tal y como se escribiría en el lenguaje C (en la práctica, un programador de C utilizaría un formato más sucinto, pero nosotros vamos a utilizar esta versión algo más larga por razones pedagógicas). Cuando se llama a esta función, esta recibe valores específicos para los parámetros formales `Radio` y `Altura` y devuelve el resultado de calcular el volumen de un cilindro con esas dimensiones. De este modo, la función podría emplearse en cualquier punto del programa mediante una sentencia como por ejemplo

```
Coste = CostePorUnidadVol * VolumenCilindro(3.45, 12.7);
```

para determinar el coste del contenido de un cilindro de radio 3.45 y altura 12.7.

Cuestiones y ejercicios

1. ¿A qué se hace referencia al hablar del "ámbito" de una variable?
2. ¿Cuál es la diferencia entre un procedimiento y una función?
3. ¿Por qué muchos lenguajes de programación implementan las operaciones de E/S como si fueran llamadas a procedimientos?
4. ¿Cuál es la diferencia entre un parámetro formal y un parámetro real?
5. Al escribir en los lenguajes de programación modernos, los programadores tienden a utilizar verbos para los nombres de los procedimientos y sustantivos para los nombres de las funciones. ¿Por qué?

6.4 Implementación de un lenguaje

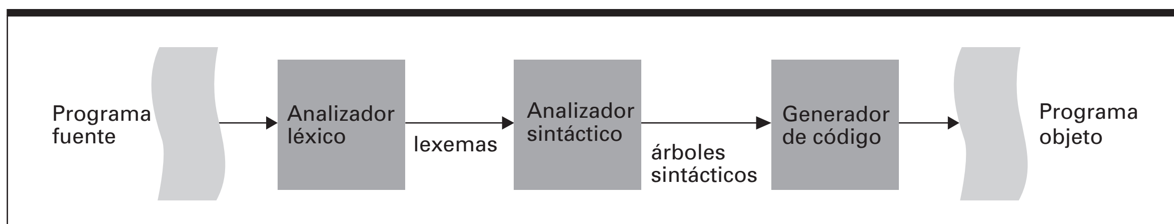
En esta sección vamos a investigar el proceso de convertir un programa escrito en un lenguaje de alto nivel a un formato ejecutable por la máquina.

El proceso de traducción

El proceso de conversión de un programa de un lenguaje a otro se denomina **traducción**. El programa en su forma original es el **programa fuente**; la versión traducida es el **programa objeto**. El proceso de traducción consta de tres actividades: análisis léxico, análisis sintáctico y generación de código, que son realizadas por unidades del traductor conocidas como **analizador léxico, analizador sintáctico** y **generador de código** (Figura 6.13).

El análisis léxico es el proceso de reconocer qué cadenas de símbolos del programa fuente representan una entidad única, o **lexema** (*token*). Por ejemplo, los tres símbolos 153 no deben interpretarse como un 1, un 5 y un 3, sino que deben reconocerse como una representación de un valor numérico. Del mismo modo, una palabra que aparezca en el programa, aunque compuesta por símbolos individuales, debe interpretarse como una sola unidad. La mayoría de las personas llevan a cabo el análisis léxico con muy poco esfuerzo consciente. Cuando nos piden que leamos en voz alta, lo que hacemos es pronunciar palabras y no caracteres individuales.

Figura 6.13 El proceso de traducción.



Por tanto, el analizador léxico lee el programa fuente símbolo a símbolo, identificando qué grupos de símbolos representan lexemas y clasificando esos lexemas según sean valores numéricos, palabras, operadores aritméticos, etc. El analizador léxico codifica cada lexema junto con la clase a la que pertenece y entrega esa información al analizador sintáctico. Durante este proceso, el analizador léxico se salta todas las sentencias de comentario.

De este modo, el analizador sintáctico ve el programa en términos de unidades léxicas (lexemas) en lugar de verlo como símbolos individuales. El trabajo del analizador sintáctico consiste en agrupar esas unidades en instrucciones. De hecho, el análisis sintáctico es el proceso de identificar la estructura gramatical del programa y reconocer el papel de cada componente. Son los detalles técnicos del análisis sintáctico los que nos hacen dudar al leer la frase

El hombre que aplastó el piano era muy alto.

(¿Quién aplastó a quién? ¿El hombre al piano o el piano al hombre?)

Para simplificar el proceso del análisis sintáctico, los primeros lenguajes de programación insistían en que cada instrucción de programa se colocara de una determinada manera en la página impresa. Tales lenguajes se conocían como **lenguajes de formato fijo**. Actualmente, la mayoría de los lenguajes de programación son **lenguajes de formato libre**, lo que significa que la colocación de las sentencias no es crítica. La ventaja de los lenguajes de formato libre radica en la capacidad del programador para organizar el programa escrito de una forma que mejore la legibilidad desde el punto de vista de los seres humanos. En estos casos, es habitual utilizar sangrados para ayudar al lector a comprender la estructura de una sentencia. En lugar de escribir

```
if Coste < EfectivoDisponible then pagar en efectivo else
usar tarjeta de crédito
```

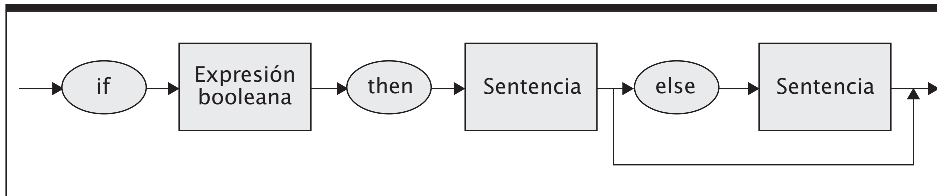
un programador podría escribir

```
if Coste < EfectivoDisponible
  then pagar en efectivo
  else usar tarjeta de crédito
```

Para que una máquina lleve a cabo el análisis sintáctico de un programa escrito en un lenguaje con formato libre, la sintaxis del lenguaje debe estar diseñada de manera que pueda identificarse la estructura del programa independientemente del espaciado utilizado en el programa fuente. Con este fin, la mayoría de los lenguajes de programación de formato libre utilizan signos de puntuación, como por ejemplo el punto y coma, para marcar el final de las sentencias, así como **palabras clave** tales como `if`, `then` y `else` para marcar el principio de cada frase individual. Estas palabras clave a menudo son **palabras reservadas**, lo que significa que el programador no puede utilizarlas para ningún otro propósito dentro del programa.

El proceso de análisis sintáctico está basado en un conjunto de reglas que definen la sintaxis del lenguaje de programación. Colectivamente, estas reglas se denominan **gramática**. Una forma de expresar estas reglas es por medio de los **diagramas sintácticos**, los cuales son representaciones gráficas de la estructura gramatical de un lenguaje. La Figura 6.14 muestra un diagrama sintáctico de la sentencia `if-then-else` de nuestro pseudocódigo del Capítulo 5. Este diagrama indica que una estructura `if-then-else` comienza con la palabra

Figura 6.14 Un diagrama sintáctico de nuestra sentencia de pseudocódigo if-then-else.



`if`, seguida de una ***Expresión booleana***, seguida por la palabra `then` y luego de una ***Sentencia***. Esta combinación puede o no ir seguida de la palabra `else` y de una ***Sentencia***. Observe que los términos que aparecen en la práctica en una sentencia `if-then-else` se han incluido dentro de la formas ovaladas, mientras que los términos que requieren una descripción más detallada, como por ejemplo ***Expresión booleana*** y ***Sentencia*** se han incluido dentro de las formas rectangulares. Estos últimos términos reciben el nombre de **no terminales**; y los términos que aparecen en los óvalos se denominan **terminales**. En una descripción completa de la sintaxis de un lenguaje, los no terminales se describen mediante diagramas adicionales.

Podemos ver un ejemplo más completo en la Figura 6.15, la cual presenta un conjunto de diagramas sintácticos que describen la sintaxis de una estructura denominada ***Expresión***, la cual trata de describir la estructura de las expresiones aritméticas simples. El primer diagrama describe una ***Expresión*** que consta de un ***Término*** que puede ir seguido o no de un símbolo + o −, al que le sigue otra ***Expresión***. El segundo diagrama describe un ***Término*** que consta solo de un ***Factor*** o bien de un ***Factor*** seguido por un símbolo × o un símbolo ÷, al que le sigue otro ***Término***. Finalmente, el último diagrama describe un ***Factor*** como uno de los símbolos `x`, `y` o `z`.

La forma en que una determinada cadena define un conjunto de diagramas sintácticos puede representarse de forma gráfica mediante un **árbol sintáctico**, como se muestra en la Figura 6.16, que ilustra el árbol sintáctico de la cadena

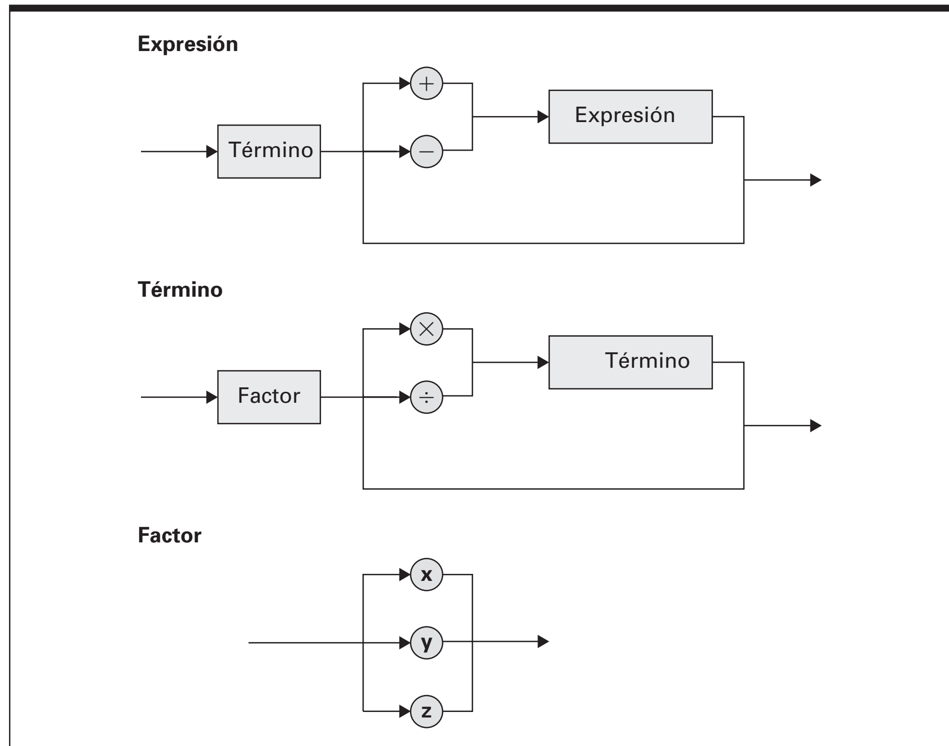
$$x + y \times z$$

basado en el conjunto de diagramas de la Figura 6.15. Observe que el árbol comienza en su parte superior con la ***Expresión*** no terminal y que cada nivel muestra cómo se descomponen los elementos no terminales de cada nivel hasta que se obtienen los símbolos de la propia cadena. En particular, la figura muestra que (de acuerdo con el primer diagrama de la Figura 6.15) una ***Expresión***

Python

Python es un lenguaje de programación que fue creado por Guido van Rossum a finales de la década de 1980. Actualmente es popular en el desarrollo de aplicaciones web, en cálculos científicos y también como lenguaje introductorio para los estudiantes. Python hace hincapié en la legibilidad e incluye elementos de los paradigmas de programación imperativo, orientado a objetos y funcional. Python es también un ejemplo de lenguaje moderno que utiliza un tipo de formato fijo. Emplea el sangrado para definir los bloques de programa, en lugar de utilizar signos de puntuación o palabras reservadas.

Figura 6.15 Diagramas sintácticos que describen la estructura de una expresión algebraica simple.



puede descomponerse como un *Término*, seguido del símbolo + y de una *Expresión*. A su vez, el *Término* puede descomponerse (utilizando el segundo diagrama de la Figura 6.15) como un *Factor* (el cual resulta ser el símbolo x), y la *Expresión* final se puede descomponer (utilizando el tercer diagrama de la Figura 6.15) como un *Término* (el cual resulta ser y × z).

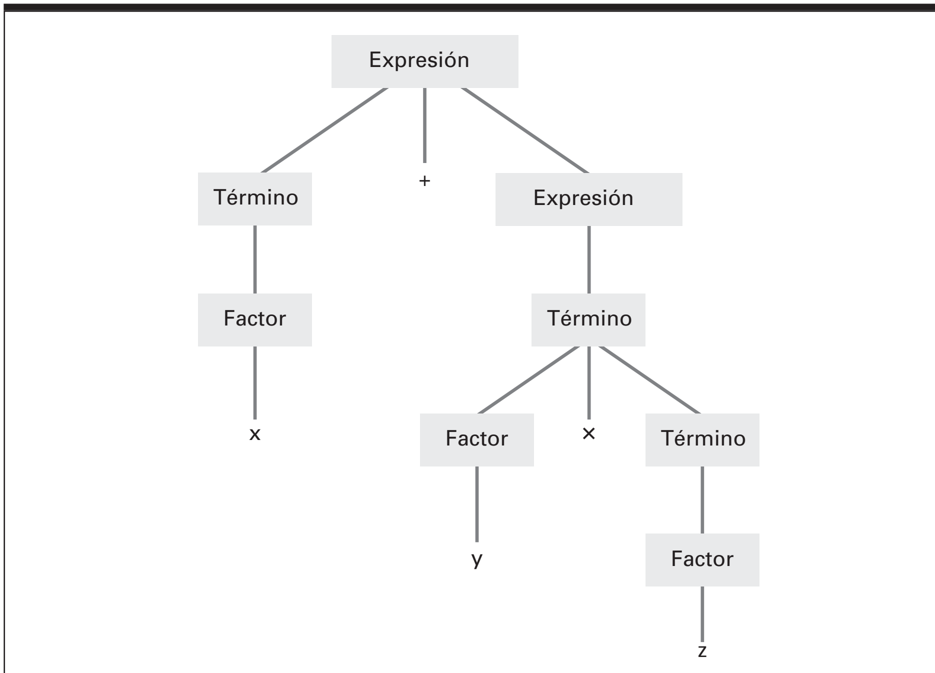
El proceso de analizar sintácticamente un programa consiste, básicamente, en construir un árbol sintáctico para el programa fuente. De hecho, un árbol sintáctico representa la interpretación que hace el analizador sintáctico de la composición gramatical del programa. Por esta razón, las reglas sintácticas que describen la estructura gramatical de un programa no deben permitir que existan dos árboles sintácticos distintos para una cadena, ya que eso llevaría a la aparición de ambigüedades dentro del analizador sintáctico. Una gramática que permita dos árboles sintácticos diferentes para una misma cadena se dice que es una **gramática ambigua**.

Las ambigüedades en las gramáticas pueden ser muy sutiles. De hecho, la regla de la Figura 6.14 contiene un error de este tipo, ya que permite los dos árboles sintácticos de la Figura 6.17 para la sentencia

```
if B1 then if B2 then S1 else S2
```

Observe que estas interpretaciones son significativamente diferentes. La primera implica que la sentencia *S2* se ejecutará si *B1* es falso; la segunda interpretación implica que *S2* se ejecutará solo si *B1* es verdadero y *B2* es falso.

Figura 6.16 El árbol de análisis sintáctico para la cadena x + y × z basado en los diagramas sintácticos de la Figura 6.15.



Las definiciones sintácticas de los lenguajes de programación formales están diseñadas para evitar estas ambigüedades. En nuestro pseudocódigo hemos evitado tales problemas mediante el uso de paréntesis. En particular, podemos escribir

```
if B1
  then (if B2 then S1)
  else S2
```

y

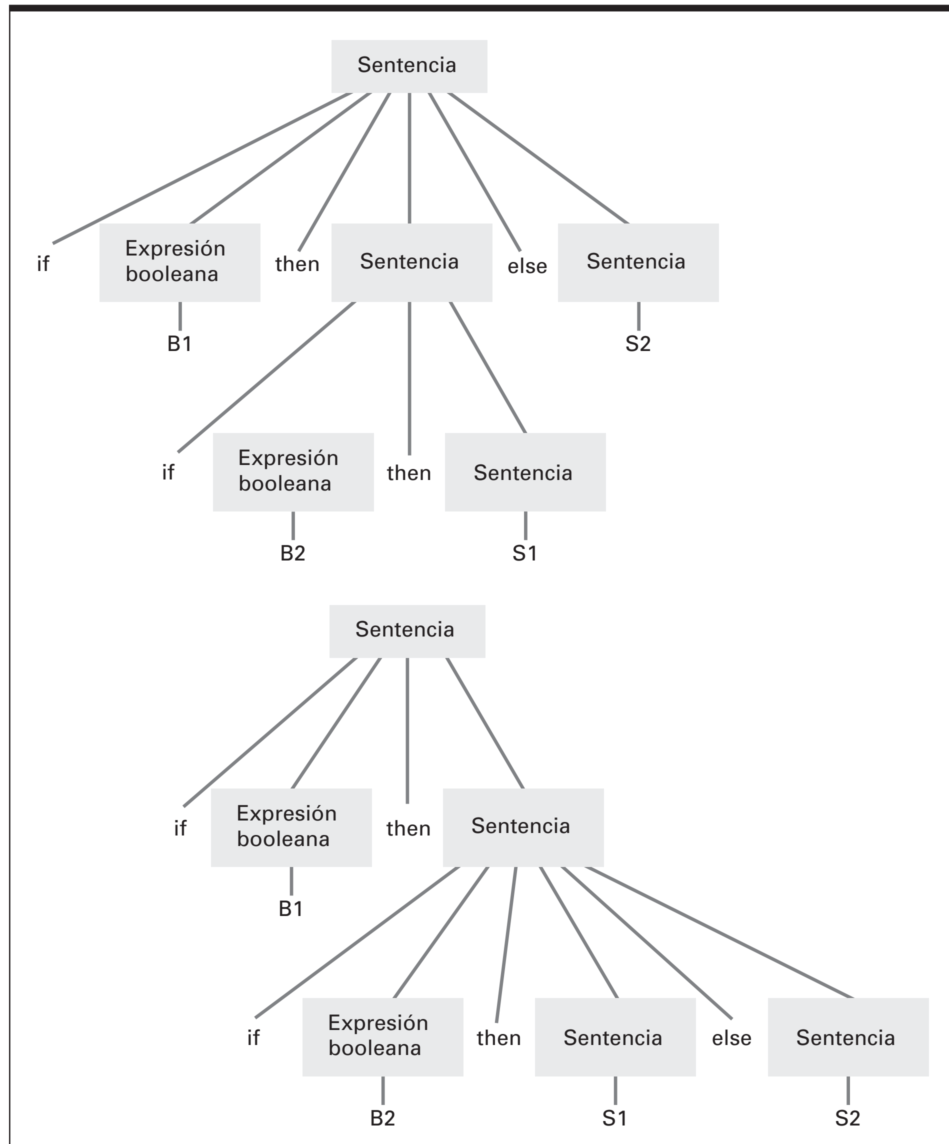
```
if B1
  then (if B2 then S1
             else S2)
```

para poder distinguir entre las dos posibles interpretaciones.

Cuando un analizador sintáctico analiza la estructura gramatical de un programa, es capaz de identificar las sentencias individuales y de diferenciar entre sentencias declarativas y sentencias imperativas. Cuando reconoce sentencias declarativas, registra la información que está siendo declarada en una tabla denominada **tabla de símbolos**. De este modo, la información que contiene la tabla de símbolos es por ejemplo los nombres de las variables que aparecen en el programa, así como los tipos de datos y las estructuras de datos asociadas con dichas variables. Entonces el analizador sintáctico se basa en esta información cuando analiza sentencias imperativas tales como

```
z ← x + y;
```

Figura 6.17 Dos árboles sintácticos distintos para la sentencia if B1 then if B2 then S1 else S2.



En particular, para determinar el significado del símbolo +, el analizador sintáctico tiene que conocer cuál es el tipo de datos asociado con x e y. Si x es de tipo float e y es de tipo char, entonces la suma de x e y no tiene ningún sentido y deberá informar de que esta instrucción es errónea. Si x e y son ambas de tipo entero, entonces el analizador sintáctico solicitará que el generador de código cree una instrucción en lenguaje máquina utilizando el código de operación que define la suma de enteros en esa máquina; si x e y fueran ambas de tipo float, entonces el analizador sintáctico solicitará que se utilice el código de operación para la suma en punto flotante; o si ambas fueran de tipo carácter, el

Implementación de Java y C#

En algunos casos, como por ejemplo en el control de una página web animada, el software debe transferirse a través de Internet y ejecutarse en máquinas remotas. Si este software se suministra en forma de código fuente, se producirán retardos adicionales en el destino debido a que el software tendrá que ser traducido al lenguaje máquina apropiado antes de ser ejecutado. Sin embargo, suministrar el software en forma de lenguaje máquina significaría que tendría que proporcionarse una versión del software diferente dependiendo del lenguaje máquina utilizado por la computadora remota.

Sun Microsystems y Microsoft han resuelto este problema diseñando "lenguajes máquina universales" (denominados bytecode en el caso de Java y .NET Common Intermediate Language en el caso de C#) a los que pueden ser traducidos los programas en código fuente. Aunque estos lenguajes no son verdaderamente lenguajes máquina, están diseñados para ser traducidos muy rápidamente. Así, si software escrito en Java o C# se traduce al "lenguaje máquina universal" apropiado, entonces puede ser transferido a otras máquinas de Internet donde podrán ser ejecutados de forma eficiente. En algunos casos, esta ejecución se realiza mediante un intérprete. En otros casos, se hace una traducción rápidamente antes de la ejecución, proceso que se conoce como **compilación just-in-time**.

analizador sintáctico podría solicitar que el generador de código creara la secuencia de instrucciones en lenguaje máquina necesaria para efectuar la operación de concatenación.

Un caso algo especial que puede surgir es si `x` es de tipo entero e `y` es de tipo float. En este caso, el concepto de suma es aplicable pero los valores no están codificados en formatos compatibles. En este caso, el analizador sintáctico puede decidir que el generador de código genere las instrucciones que permitan convertir un valor al tipo de datos del otro valor y luego llevar a cabo la suma. Este tipo de conversión implícita entre tipos de datos se denomina **coerción**.

Muchos diseñadores de lenguajes de programación no ven con buenos ojos la coerción, porque puede alterar el valor de los datos, dando lugar a sutiles errores en el programa. Argumentan que la necesidad de usar la coerción normalmente indica un fallo en el diseño del programa que no debería ser admitido por el analizador sintáctico. El resultado es que la mayor parte de los lenguajes modernos son **fuertemente tipados**, lo que significa que todas las actividades solicitadas por un programa deben implicar datos de tipos compatibles. Algunos lenguajes, como por ejemplo Java, permiten la conversión de tipos implícita siempre que se trate de una **promoción de tipos**, es decir, cuando se convierte un valor de poca precisión en un valor de una precisión mayor. Las conversiones de tipos implícitas que pueden modificar un valor son consideradas como errores. En la mayoría de los casos, un programador puede solicitar este tipo de conversiones mediante una **conversión de tipos explícita**, que indica al compilador que el programador es consciente de que se va a aplicar una conversión de tipos.

La última actividad del proceso de traducción es la **generación del código**, que es el proceso de construir las instrucciones en lenguaje máquina para implementar las sentencias que ha reconocido el analizador sintáctico. Este

proceso presenta numerosos problemas, siendo uno de ellos el de generar versiones eficientes de los programas en lenguaje máquina. Por ejemplo, considere la tarea de traducir la siguiente secuencia de dos sentencias

```
x ← y + z;
w ← x + z;
```

Si estas sentencias se traducen como sentencias individuales, cada una de ellas requerirá que los datos sean transferidos desde la memoria principal al procesador antes de que la suma indicada se realice. Sin embargo, puede conseguirse una mayor eficiencia dándose cuenta de que una vez que se ha ejecutado la primera sentencia, los valores de `x` y `z` ya están en los registros de propósito general del procesador y que, por tanto, no es necesario cargarlos desde la memoria principal antes de efectuar la segunda suma. Las consideraciones de implementación de este estilo se denominan **optimización de código** y constituyen una tarea importante del generador de código.

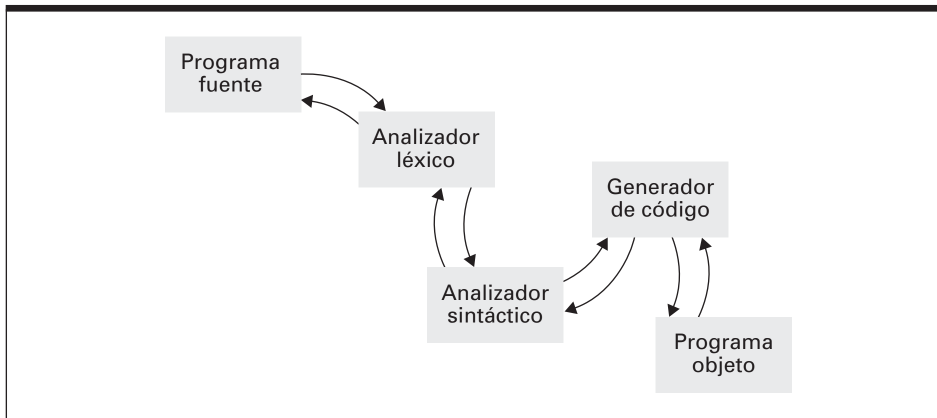
Por último, debemos observar que el análisis léxico, el análisis sintáctico y la generación de código no son pasos que se realicen en un orden secuencial estricto, sino que estas actividades se entrelazan. El analizador léxico comienza leyendo los caracteres del programa fuente hasta identificar el primer lexema, momento en el que lo pasa al analizador sintáctico. Cada vez que el analizador sintáctico recibe un lexema procedente del analizador léxico, analiza la estructura gramatical que está siendo leída. En esta situación puede bien solicitar más lexemas al analizador léxico o, si reconoce que ya se ha leído una frase o instrucción completa, llamar al generador de código para producir las instrucciones en lenguaje máquina apropiadas. Cada solicitud de este tipo hace que el generador de código cree instrucciones máquina que son añadidas al programa objeto. Además, la tarea de traducir un programa de un lenguaje a otro cumple de forma natural el paradigma de la orientación a objetos. El programa fuente, el analizador léxico, el analizador sintáctico, el generador de código y el programa objeto son objetos que interactúan intercambiándose mensajes a medida que cada objeto va realizando su tarea (Figura 6.18).

Paquetes de desarrollo software

Las herramientas software tales como los editores y los traductores utilizados en el proceso de desarrollo software a menudo se agrupan en un paquete que funciona como un sistema de desarrollo software integrado. Este tipo de sistemas se clasificaría como software de aplicación dentro del esquema de clasificación que hemos definido en la Sección 3.2. Con un paquete de aplicación de este tipo, un programador tendrá acceso a un editor para escribir los programas, a un traductor para convertir los programas a lenguaje máquina y a una serie de herramientas de depuración que le permitirán seguir la ejecución de un programa erróneo con el fin de descubrir en qué punto se desvía del funcionamiento correcto.

Son muchas las ventajas de utilizar estos sistemas integrados. Quizá la más evidente sea que el programador puede pasar del editor a las herramientas de depuración muy fácilmente, a medida que realiza y prueba los cambios en el programa. Además, muchos paquetes de desarrollo software permiten enlazar las unidades de programa relacionadas que se están desarrollando de tal forma

Figura 6.18 Una técnica orientada a objetos para el proceso de traducción.



que se simplifica el acceso a dichas unidades relacionadas. Algunos paquetes mantienen registros que informan de qué unidades de programa de un cierto grupo de unidades relacionadas han sido modificadas desde que se realizó la última prueba. Estas capacidades son muy ventajosas en el desarrollo de sistemas software grandes en los que diferentes programadores desarrollan muchas unidades interrelacionadas.

A menor escala, los editores de los paquetes de desarrollo software suelen estar personalizados para el lenguaje que se esté utilizando. Normalmente, un editor de este tipo proporcionará el sangrado de línea automático que sea el estándar de facto para el lenguaje en cuestión y, en algunos casos, es posible que reconozca y complete automáticamente las palabras clave después de que el programador haya escrito los primeros caracteres. Además, es posible que el editor resalte las palabras clave dentro del programa fuente (quizá con un color), haciendo que los programas sean más fáciles de leer.

En el siguiente capítulo veremos que los desarrolladores de software buscan incesantemente formas mediante las que construir nuevos sistemas software a partir de bloques prefabricados denominados componentes, lo que conduce a un nuevo modelo de desarrollo software denominado arquitectura basada en componentes. Los paquetes de desarrollo software basados en el modelo de arquitectura de componentes a menudo utilizan interfaces gráficas en las que los componentes pueden representarse mediante iconos en la pantalla de la computadora. Con esta configuración, los programadores (o ensambladores de componentes) seleccionan los componentes deseados con el ratón. Entonces, un componente seleccionado puede personalizarse utilizando el editor del paquete y luego asociarse a los demás componentes apuntando y haciendo clic con el ratón. Estos paquetes representan un importante avance en la búsqueda de mejores herramientas de desarrollo software.

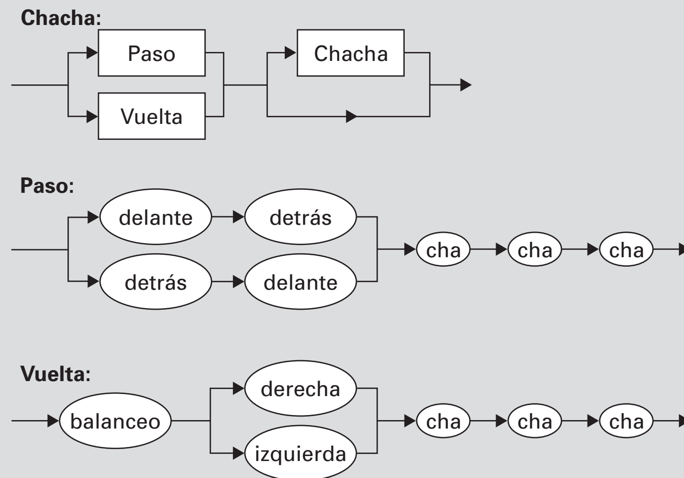
Cuestiones y ejercicios

1. Describa los tres pasos principales del proceso de traducción.
2. ¿Qué es una tabla de símbolos?

3. ¿Cuál es la diferencia entre un elemento terminal y uno no terminal?
4. Dibuje el árbol sintáctico de la expresión

$$x \times y + x + z$$

basándose en los diagramas sintácticos de la Figura 6.15.
5. Describa las cadenas que componen la estructura Chacha según los siguientes diagramas sintácticos.



6.5 Programación orientada a objetos

En la Sección 6.1 hemos visto que el paradigma de orientación a objetos conlleva el desarrollo de unidades de programa activas denominadas **objetos**, cada uno de los cuales contiene procedimientos que describen cómo dicho objeto debe responder a los diversos estímulos. Aplicar la técnica de orientación a objetos a un problema consiste en identificar y describir los objetos como unidades autocontenidas. A su vez, los lenguajes de programación orientados a objetos proporcionan sentencias que permiten describir tanto a los objetos como su comportamiento. En esta sección, presentaremos algunas de estas sentencias tal y como aparecen en los lenguajes C++, Java y C#, que son tres de los lenguajes de orientación a objetos más destacados y utilizados actualmente.

Clases y objetos

Consideremos la tarea de desarrollar un sencillo juego de computadora en el que la misión del jugador es proteger la Tierra de los meteoritos que caen sobre ella disparándoles con láseres de alta potencia. Cada láser dispone de una fuente de energía interna finita que se consume parcialmente cada vez que se dispara el láser. Una vez que esta fuente de energía se agota, el láser se vuelve inutilizable. Cada láser tiene que ser capaz de responder a las órdenes de apuntar más a la derecha, apuntar más a la izquierda y disparar su rayo láser.

En el paradigma de orientación a objetos, cada láser del juego para computadora podría implementarse como un objeto que contiene un registro de la potencia que queda por consumir, así como procedimientos para modificar dónde apuntar y para disparar el rayo láser. Puesto que todos los objetos láser tienen las mismas propiedades, pueden describirse mediante una plantilla común. En el paradigma de orientación a objetos, una plantilla para una colección de objetos se dice que es una **clase**.

En el Capítulo 8 veremos las similitudes entre clases y tipos de datos. Por el momento, fijémonos simplemente en que una clase describe las características comunes de una colección de objetos, de forma similar a como el concepto de tipo de datos primitivo entero se asocia con las características comunes de números como el 1, el 5 y el 82. Una vez que un programador ha incluido la descripción de una clase en un programa, dicha plantilla se puede utilizar para construir y manipular objetos de ese "tipo", de la misma forma que el tipo primitivo de enteros permite la manipulación de "objetos" de tipo entero.

En los lenguajes C++, Java y C# una clase se describe mediante una sentencia de la forma

```
class Nombre
{
     .
     .
     .
}
```

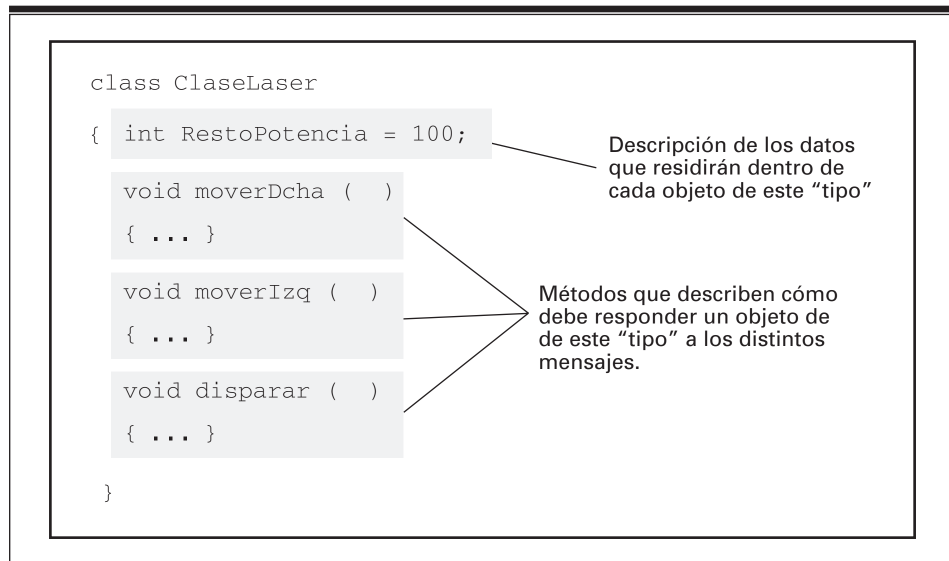
donde *Nombre* es el nombre mediante el que la clase puede ser referenciada en cualquier punto del programa. Dentro de las llaves se describen las propiedades de la clase. En particular, una clase denominada `ClaseLaser` que describa la estructura de un láser para nuestro juego de computadora se ilustra en la Figura 6.19. La clase consta de la declaración de una variable denominada `RestoPotencia` (de tipo entero) y tres procedimientos denominados `moverDcha`, `moverIzq` y `disparar`. Estos procedimientos describen las rutinas que deben ejecutarse para llevar a cabo la acción correspondiente. Así, cualquier objeto que se construya a partir de esta plantilla tendrá estas características: una variable denominada `RestoPotencia` y tres procedimientos de nombre `moverDcha`, `moverIzq` y `disparar`.

Una variable que reside dentro de un objeto, como por ejemplo `RestoPotencia`, se dice que es una **variable de instancia** y los procedimientos asociados a un objeto se denominan **métodos** (o funciones miembro en el lenguaje C++). Observe que en la Figura 6.19 la variable de instancia `RestoPotencia` se describe utilizando una sentencia de declaración similar a las que hemos visto en la Sección 6.2 y que los métodos se describen de forma similar a los procedimientos y funciones que hemos visto en la Sección 6.3. Después de todo, las declaraciones de variables de instancia y las descripciones de los métodos son, básicamente, conceptos relativos a la programación imperativa.

Una vez que hemos descrito la clase `ClaseLaser` en nuestro programa del juego, podemos declarar tres variables `Laser1`, `Laser2` y `Laser3` para que sean de "tipo" `ClaseLaser` mediante una sentencia de la forma

```
ClaseLaser Laser1, Laser2, Laser3;
```

Figura 6.19 Estructura de una clase que describe una pistola láser en un juego de computadora.



Observe que esta sentencia tiene el mismo formato que la sentencia

```
int x, y, z;
```

que utilizaríamos para declarar tres variables de nombres `x`, `y` y `z` de tipo entero (`int`), como hemos visto anteriormente en la Sección 6.2. Ambas sentencias constan del nombre de un "tipo" seguido de una lista de las variables que se desea declarar. La diferencia está en que la segunda sentencia establece que las variables `x`, `y` y `z` se utilizarán en el programa para hacer referencia a los elementos de tipo entero (que es un tipo primitivo), mientras que la primera sentencia establece que las variables `Laser1`, `Laser2` y `Laser3` se emplearán en el programa para hacer referencia a los elementos de "tipo" `ClaseLaser` (que es un "tipo" definido dentro del programa).

Una vez que hemos declarado las variables `Laser1`, `Laser2` y `Laser3` como del "tipo" `ClaseLaser`, podemos asignarles valores. En este caso, los valores tienen que ser objetos que se ajusten al "tipo" `ClaseLaser`. Estas asignaciones se pueden realizar mediante sentencias de asignación, aunque a menudo es conveniente asignar los valores iniciales a las variables dentro de las propias sentencias de asignación utilizadas para declararlas. En el lenguaje C++, la asignación de valores iniciales se hace de forma automática en las sentencias de declaración. Es decir, la sentencia

```
ClaseLaser Laser1, Laser2, Laser3;
```

no solo define las variables `Laser1`, `Laser2` y `Laser3`, sino que también crea tres objetos de "tipo" `ClaseLaser`, siendo cada uno de ellos el valor correspondiente a cada una de las variables. En los lenguajes Java y C#, estas asignaciones iniciales se hacen de la misma forma que las asignaciones iniciales a las variables de tipos primitivos. En particular, en tanto que la sentencia

```
int x = 3;
```

no solo declara x como una variable de tipo entero sino que también le asigna el valor 3, la sentencia

```
ClaseLaser Laser1 = new ClaseLaser();
```

declara la variable Laser1 como de "tipo" ClaseLaser y también crea un nuevo objeto utilizando la plantilla ClaseLaser que asigna dicho objeto como el valor inicial de Laser1.

Llegados a este punto, vamos a hacer una pausa para hacer hincapié en la diferencia entre clase y objeto. Una clase es una plantilla a partir de la cual se construyen objetos. Una clase se puede usar para crear numerosos objetos. A menudo nos referiremos a un objeto diciendo que es una **instancia** de la clase a partir de la que ha sido construido. Por tanto, en nuestro juego de computadora, Laser1, Laser2 y Laser3 son variables cuyos valores son instancias de la clase ClaseLaser.

Después de utilizar las sentencias declarativas para crear las variables Laser1, Laser2 y Laser3 y asignarles objetos, podemos continuar con nuestro programa del juego escribiendo sentencias imperativas que activen los métodos apropiados dentro de esos objetos (en el lenguaje de programación orientada a objetos, esto se denomina enviar mensajes a los objetos). En particular, podríamos hacer que el objeto asignado a la variable Laser1 ejecutara su método disparar mediante la sentencia

```
Laser1.disparar();
```

O podríamos hacer que el objeto asignado a Laser2 ejecutara su método moverIzq mediante la sentencia

```
Laser2.moverIzq();
```

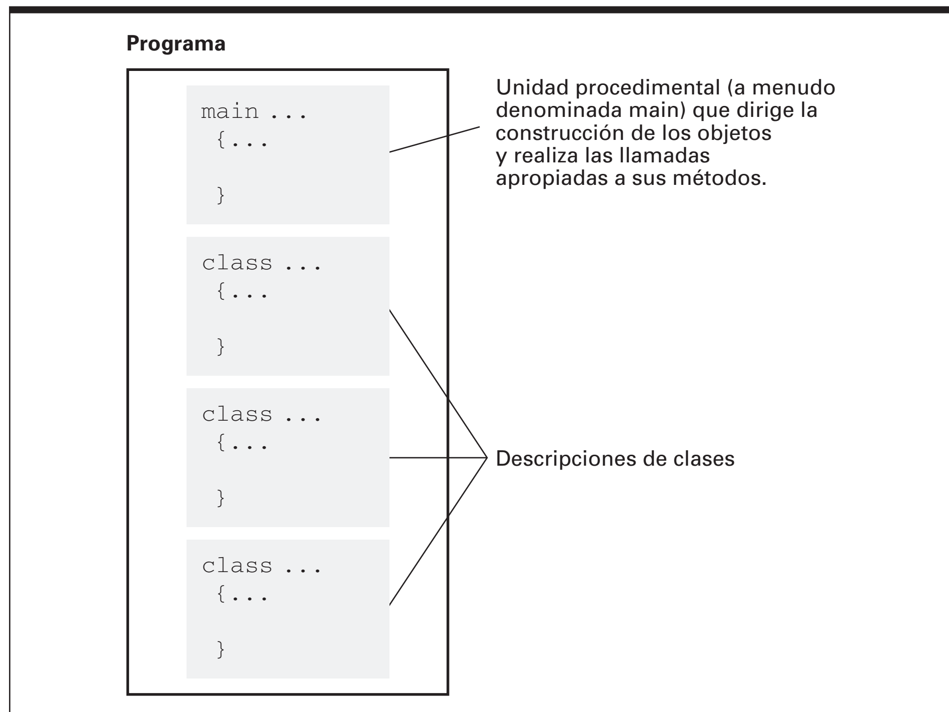
Realmente, estas sentencias son poco más que llamadas a procedimiento. De hecho, la primera de ellas es una llamada al procedimiento (el método) disparar del objeto asignado a la variable Laser1, y la segunda es una llamada al procedimiento moverIzq del objeto asignado a la variable Laser2.

Hasta aquí, nuestro ejemplo del juego de los meteoritos nos ha proporcionado las bases para comprender la estructura global de un programa orientado a objetos típico (Figura 6.20). Un programa de este tipo contendrá una variedad de descripciones de clases similares a las de la Figura 6.19, describiendo cada una de ellas la estructura de uno o más de los objetos utilizados en el programa. Además, el programa contendrá un segmento de programa imperativo (normalmente asociado con el nombre "main") que detallará la secuencia de pasos que hay llevar a cabo inicialmente cuando se ejecute el programa. Este segmento contendrá sentencias de declaración similares a nuestras declaraciones de los láseres, con el fin de definir los objetos empleados en el programa, así como sentencias imperativas que llamarán a los métodos de dichos objetos para que se ejecuten.

Constructores

Una vez que se ha construido un objeto, suele ser necesario llevar a cabo algunas actividades para personalizarlo. Por ejemplo, en nuestro juego de computadora de los meteoritos podemos querer que los distintos láseres tengan potencias iniciales distintas, lo que quiere decir que las variables de instancia

Figura 6.20 Estructura de un programa orientado a objetos típico.



denominadas `RestoPotencia` de los diversos objetos deberán tener valores iniciales diferentes. Estas necesidades de inicialización se realizan definiendo métodos especiales, denominados **constructores**, dentro de la clase apropiada. Los constructores se ejecutan automáticamente cuando se construye un objeto a partir de la clase. Un constructor se identifica dentro de una definición de clase por el hecho de que es un método con el mismo nombre que la clase.

La Figura 6.21 presenta una extensión de la definición original de `ClaseLaser` mostrada en la Figura 6.19. Observe que contiene un constructor en la forma de un método de nombre `ClaseLaser`. Este método asigna a la variable de instancia `RestoPotencia` el valor que recibe como parámetro. Por tanto, cuando un objeto se construye a partir de esta clase, este método se ejecutará, haciendo que `RestoPotencia` se inicialice con el valor apropiado.

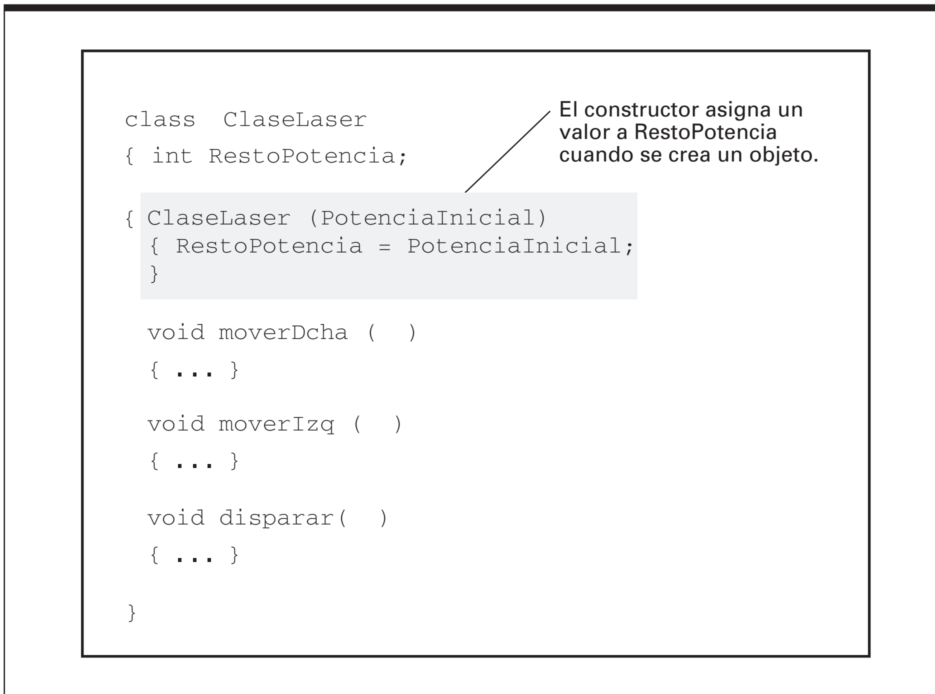
Los parámetros reales utilizados por un constructor se identifican en la lista de parámetros de la sentencia que lleva a cabo la creación del objeto. Por tanto, basándonos en la definición de clase dada en la Figura 6.21, un programador de C++ podría escribir

```
ClaseLaser Laser1(50), Laser2(100);
```

para crear dos objetos de tipo `ClaseLaser`, uno con el nombre de `Laser1` y una reserva de potencia inicial de 50 y el otro con el nombre de `Laser2` y una reserva de potencia inicial de 100. Los programadores en Java y C# podrían llevar a cabo la misma tarea con las sentencias

```
ClaseLaser Laser1 = new ClaseLaser (50);
ClaseLaser Laser2 = new ClaseLaser (100);
```

Figura 6.21 Una clase con un constructor.



Características adicionales

Supongamos ahora que deseamos mejorar nuestro juego para computadora de los meteoritos de manera que un jugador que alcance una determinada puntuación sea premiado con la recarga hasta la potencia inicialmente configurada de algunos de los láseres. Estos láseres tendrán las mismas propiedades que los láseres anteriores y además serán recargables.

Para simplificar la descripción de los objetos con características similares pero con ciertas diferencias, los lenguajes orientados a objetos permiten que una clase adquiera las propiedades de otra por medio de una técnica conocida como **herencia**. Por ejemplo, suponga que estamos utilizando Java para desarrollar nuestro juego. Podríamos utilizar en primer lugar la sentencia `class` descrita anteriormente para definir una clase de nombre `ClaseLaser` que describa aquellas propiedades que son comunes a todos los láseres del programa. Y después podríamos emplear la sentencia

```
class LaserRecargable extends ClaseLaser
{
      .
      .
      .
}
```

para describir otra clase denominada `LaserRecargable`. (Los programadores de C++ y C# simplemente tendrán que sustituir la palabra `extends` por un signo de dos puntos.) Aquí, la cláusula `extends` indica que esta clase tiene las características que herede de la clase `ClaseLaser`, así como aquellas caracte-

rísticas que se especifiquen dentro de las llaves. En nuestro caso, estas llaves podrían contener un nuevo método (quizá denominado `recarga`) que describiría los pasos necesarios para restablecer la variable de instancia `RestoPotencia` a su valor original. Una vez que estas clases estuvieran definidas, podríamos utilizar la sentencia

```
ClaseLaser Laser1, Laser2;
```

para declarar que `Laser1` y `Laser2` son variables que hacen referencia a los láseres tradicionales, y la sentencia

```
LaserRecargable Laser3, Laser4;
```

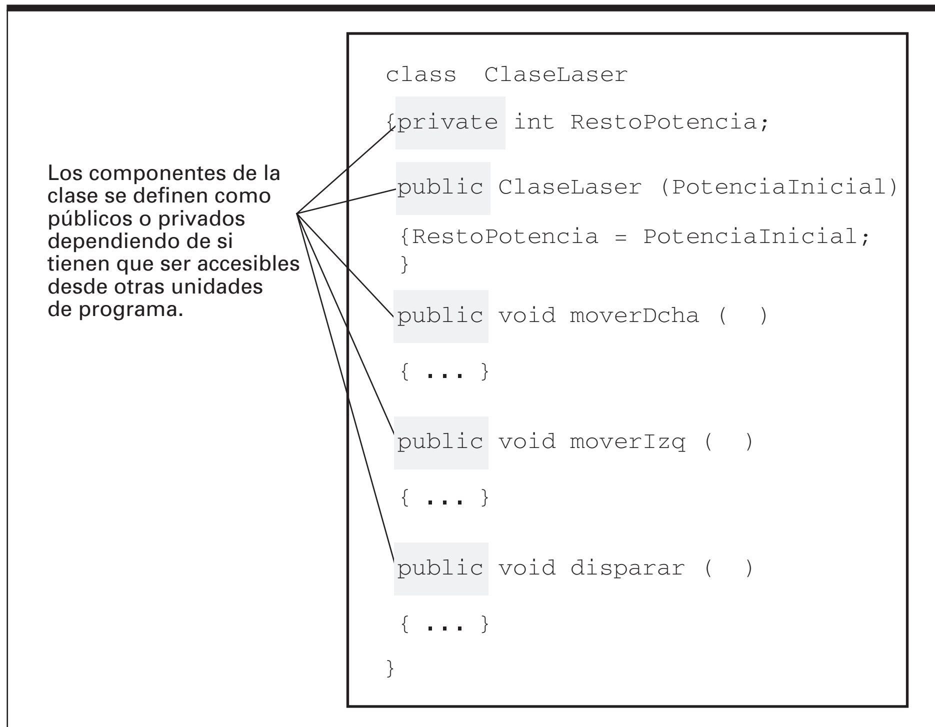
para declarar `Laser3` y `Laser4` como variables que hacen referencia a los láseres que tienen las propiedades adicionales descritas en la clase `LaserRecargable`.

El uso de la herencia lleva a la existencia de una variedad de objetos con características similares pero con algunas diferencias, lo que a su vez conduce a un fenómeno reminiscente de sobrecarga, que ya vimos en la Sección 6.2 (recuerde que la sobrecarga hace referencia al uso de un mismo símbolo, como por ejemplo el símbolo `+`, para representar operaciones diferentes dependiendo del tipo de datos de los operandos). Suponga que un paquete gráfico orientado a objetos consta de una serie de objetos, representando cada uno de ellos una forma (círculo, rectángulo, triángulo, etc.). Una determinada imagen puede estar formada por un conjunto de estos objetos. Cada objeto "conoce" su tamaño, posición y color, y también sabe cómo responder a los mensajes que le llegan; por ejemplo, desplazarse a una nueva posición o dibujarse en la pantalla. Para dibujar una imagen, nosotros simplemente enviamos el mensaje "dibújate" a cada objeto de la imagen. Sin embargo, la rutina utilizada para dibujar un objeto varía dependiendo de cuál sea la forma del objeto (dibujar un cuadrado no sigue el mismo proceso que dibujar un círculo). Esta interpretación personalizada de un mensaje es lo que se denomina **polimorfismo**; y se dice que el mensaje es polimórfico.

Otra característica asociada con la programación orientada a objetos es la **encapsulación**, la cual hace referencia a restringir el acceso a las propiedades internas de un objeto. Decir que ciertas características de un objeto están *encapsuladas* significa que solo el propio objeto es capaz de acceder a ellas. Las características que están encapsuladas se dice que son privadas. Las características que son accesibles desde el exterior del objeto se dice que son públicas.

Por ejemplo, volvamos sobre nuestra clase `ClaseLaser` inicialmente definida en la Figura 6.19. Recuerde que describía una variable de instancia `RestoPotencia` y tres métodos (`moverDcha`, `moverIzq` y `disparar`). Otras unidades del programa pueden acceder a estos métodos para conseguir que una instancia de `ClaseLaser` lleve a cabo la acción apropiada. Pero el valor de `RestoPotencia` solo debería poder ser modificado por los métodos internos de la instancia; ninguna otra unidad de programa debe poder acceder directamente a este valor. Para cumplir estas reglas tenemos que definir `RestoPotencia` como privada y los métodos `moverDcha`, `moverIzq` y `disparar` como públicos, como se muestra en la Figura 6.22. Una vez incluidas estas definiciones, cualquier intento de acceder al valor de `RestoPotencia` desde fuera del objeto en el que reside se identificará como un error en el momento en que se traduzca el programa, lo que forzará al programador a corregir el problema antes de continuar.

Figura 6.22 Nuestra definición de ClaseLaser utilizando encapsulación como aparecería en un programa en Java o C#.



Cuestiones y ejercicios

1. ¿Cuál es la diferencia entre un objeto y una clase?
2. ¿Qué clases de objetos distintos de `ClaseLaser` pueden encontrarse en el ejemplo del juego de computadora utilizado en esta sección? ¿Qué variables de instancia además de `RestoPotencia` podrían encontrarse en la clase `ClaseLaser`?
3. Suponga que las clases `EmpleadoTiempoParcial` y `EmpleadoTiempoCompleto` heredan las propiedades de la clase `Empleado`. ¿Cuáles son algunas de las características que esperaría encontrar en cada clase?
4. ¿Qué es un constructor?
5. ¿Por qué algunos de los elementos de una clase se definen como privados?

6.6 Programación de actividades concurrentes

Suponga que nos piden diseñar un programa para generar una animación para un juego de acción para computadora en el que aparecieran múltiples naves espaciales enemigas atacando. Una posible solución consistiría en diseñar un

único programa que controlara toda la pantalla de animación. Dicho programa se encargaría de dibujar cada una de las naves espaciales, lo cual (si la animación tiene que parecer suficientemente realista) significaría que el programa tendría que controlar las características individuales de numerosas naves. Otra solución alternativa sería diseñar un programa para controlar la animación de un única nave espacial, cuyas características se determinaran mediante parámetros asignados al principio de la ejecución del programa. Entonces, la animación completa podría construirse creando múltiples activaciones de este programa, cada una con su propio conjunto de parámetros. Ejecutando estas activaciones simultáneamente, podríamos obtener la ilusión de que hay muchas naves espaciales individuales moviéndose al mismo tiempo por la pantalla.

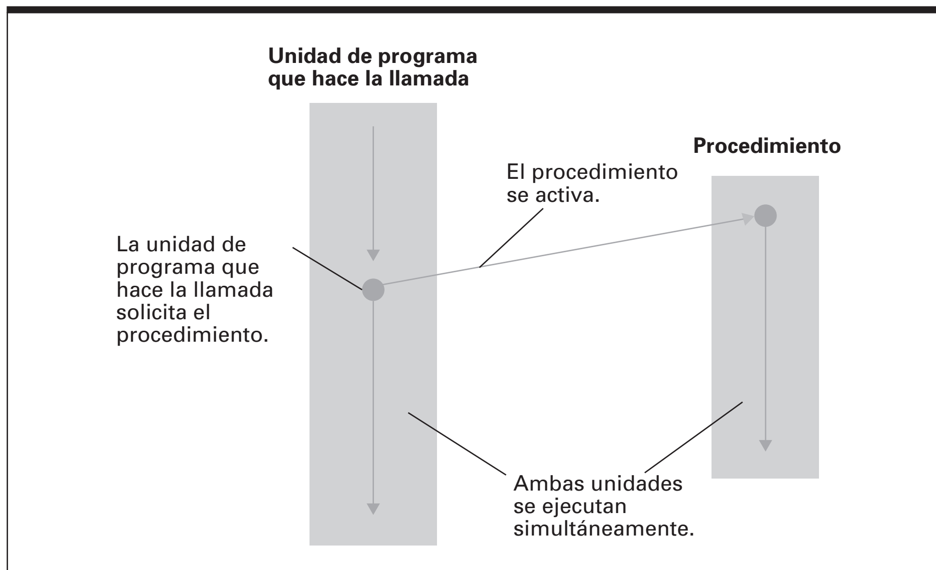
Este tipo de ejecución simultánea de múltiples activaciones se denomina **procesamiento paralelo** o **procesamiento concurrente**. El verdadero procesamiento paralelo requiere disponer de varios procesadores, cada uno de los cuales se encarga de ejecutar una activación. Cuando solo hay disponible un procesador, la ilusión del procesamiento paralelo se obtiene permitiendo que las activaciones compartan el tiempo de un único procesador, de manera similar a la forma en que se implementan los sistemas de multiprogramación (Capítulo 3).

Muchas aplicaciones modernas para computadora se suelen resolver más fácilmente en un contexto de procesamiento paralelo que en el contexto más tradicional formado por una única secuencia de sentencias. A su vez, los lenguajes de programación más recientes proporcionan la sintaxis para expresar las estructuras semánticas requeridas para la computación en paralelo. El diseño de un lenguaje de este tipo requiere identificar esas estructuras semánticas y desarrollar una sintaxis para representarlas.

Cada lenguaje de programación tiende a implementar el paradigma de procesamiento paralelo desde su propio punto de vista, lo que da como resultado la existencia de diferentes terminologías. Por ejemplo, lo que hemos denominado de manera informal "activación" se denomina tarea en la jerga de Ada e hilo en Java. Es decir, en un programa Ada, las acciones simultáneas se realizan creando múltiples ***tareas***, mientras que en Java lo que se crean son múltiples ***hilos***. En cualquier caso, el resultado es que se generan y ejecutan múltiples actividades, de forma bastante similar a como funcionan los procesos bajo el control de un sistema operativo multitarea. Vamos a adoptar la terminología de Java y nos referiremos a esos "procesos" como hilos.

Quizá la acción más básica que es preciso expresar en un programa que implique procesamiento paralelo es, precisamente, la de crear nuevos hilos. Si queremos ejecutar al mismo tiempo múltiples activaciones del programa de la nave espacial, necesitaremos una sintaxis para poder dar esa orden. Esa generación de nuevos hilos es similar a la de solicitar la ejecución de un procedimiento tradicional. La diferencia es que en un entorno tradicional, la unidad de programa que solicita la activación de un procedimiento no continúa su procesamiento hasta que el procedimiento solicitado termina (recuerde la Figura 6.8), mientras que en el contexto de procesamiento paralelo la unidad de programa solicitante continúa su ejecución mientras que el procedimiento solicitado realiza su tarea (Figura 6.23). Por tanto, para crear múltiples naves espaciales que se desplacen por la pantalla, tendríamos que escribir un programa principal que simplemente generara múltiples activaciones del programa de la nave espacial, a cada una de las cuales le propor-

Figura 6.23 Generación de hilos.



cionaría los parámetros que describen las características distintivas de esa nave espacial.

Un problema más complejo asociado con el procesamiento paralelo es el relacionado con la comunicación entre hilos. Por ejemplo, en nuestro caso de la nave espacial, los hilos que representan a las diferentes naves espaciales pueden necesitar comunicarse entre ellos sus ubicaciones para poder coordinar sus actividades. En otros casos, un hilo podría necesitar esperar a que otro hilo alcance un cierto punto dentro de su secuencia de cálculo; o bien, un hilo podría tener que detener la ejecución de otro, hasta que el primero haya completado una tarea determinada.

Estas necesidades de comunicación han sido objeto de estudio durante mucho tiempo entre los expertos en Ciencias de la computación, y muchos lenguajes de programación recientes reflejan diversos enfoques para la solución de los problemas de interacción entre hilos. Por ejemplo, consideremos los problemas de comunicación que nos encontramos cuando dos hilos manipulan los mismos datos (este ejemplo se presenta con más detalle en la Sección opcional 3.4). Si cada una de los dos hilos que se están ejecutando concurrentemente necesitan sumar el valor tres a un elemento de datos común, hace falta algún método para garantizar que uno de los hilos pueda completar su transacción antes de que a la otra se le permita realizar su tarea. En caso contrario, ambas podrían iniciar sus cálculos individuales con el mismo valor inicial, lo que querría decir que el resultado final solo se incrementaría en tres en lugar de en seis. Cuando a un dato solo puede acceder a un hilo en cada momento, decimos que ese dato tiene acceso mutuamente exclusivo.

Una forma de implementar el acceso mutuamente exclusivo consiste en escribir unidades de programa que describan los hilos implicados, de modo que cuando un hilo esté utilizando datos compartidos, bloquee el acceso de otros hilos a dichos datos hasta que ese acceso sea seguro (este es el enfoque descrito

Programación de teléfonos inteligentes

El software para dispositivos portátiles, móviles e integrados a menudo se desarrolla utilizando los mismos lenguajes de programación de propósito general que se emplean en otros contextos. Con un teclado de mayor tamaño y una buena dosis de paciencia, pueden escribirse aplicaciones para teléfonos inteligentes utilizando el propio teléfono. Sin embargo, en la mayoría de los casos, el software para los teléfonos inteligentes se desarrolla en computadoras de sobremesa empleando sistemas software especiales que proporcionan herramientas para la edición, traducción y prueba del software para teléfonos inteligentes. Con frecuencia, aplicaciones simples se escriben en Java, C++ y C#. No obstante, escribir aplicaciones más complejas o software del núcleo del sistema puede requerir soporte adicional para la programación concurrente y la programación dirigida por eventos.

en la Sección opcional 3.4, en la que hemos designado con el nombre de región crítica a aquella parte de un proceso que accede a los datos compartidos). La experiencia demuestra que esta solución tiene la desventaja de distribuir la tarea de garantizar la exclusión mutua entre varias partes de programa; cada unidad de programa que accede a los datos debe ser diseñada apropiadamente para imponer esa exclusión mutua, por lo que un error en un único segmento puede corromper todo el sistema. Por esta razón, muchos expertos argumentan que una mejor solución consiste en dotar al propio elemento de datos de la capacidad para controlar el acceso a sí mismo. En resumen, en lugar de confiar en que los hilos que acceden a los datos se protejan frente a los accesos múltiples, se asigna al propio elemento de datos esta responsabilidad. El resultado es que el control de acceso está concentrado en un único punto del programa, en lugar de estar disperso entre múltiples unidades de programa. Un elemento de datos ampliado con la capacidad de controlar el acceso a sí mismo, se suele denominar **monitor**.

En conclusión, el diseño de lenguajes de programación para el procesamiento paralelo implica desarrollar formas de expresar cosas tales como la creación de hilos, la detención y la reanudación de hilos, la identificación de regiones críticas y la composición de monitores.

Para terminar, debemos recalcar que aunque la animación proporciona un entorno interesante en el que explorar las cuestiones de la computación paralela, es únicamente uno de entre muchos campos distintos que se benefician de las técnicas de procesamiento paralelo. Otras áreas de aplicación incluyen la predicción meteorológica, el control del tráfico aéreo, la simulación de sistemas complejos (desde reacciones nucleares a tráfico de peatones), redes de computadoras y mantenimiento de bases de datos.

Cuestiones y ejercicios

1. ¿Podría citar algunas propiedades que pueden encontrarse en un lenguaje de programación para procesamiento concurrente y que no estén presentes en un lenguaje más tradicional?

2. Describa dos métodos que garanticen el acceso mutuamente exclusivo a los datos.
3. Identifique algunos entornos distintos del de la animación en los que resulte beneficiosa la computación en paralelo.

6.7 Programación declarativa

En la Sección 6.1 hemos dicho que la lógica formal proporciona un algoritmo general de resolución de problemas alrededor del cual podemos construir un sistema de programación declarativa. En esta sección vamos a investigar más a fondo esta afirmación, presentando primero los rudimentos del algoritmo y luego examinando brevemente un lenguaje de programación declarativo basado en él.

Deducción lógica

Suponga que sabemos que la rana Gustavo está actuando o que la rana Gustavo está enferma y suponga también que nos dicen que la rana Gustavo no está actuando. Entonces podríamos concluir que la rana Gustavo debe estar enferma. Este es un ejemplo de un principio de razonamiento deductivo denominado **resolución**. La resolución es una de entre múltiples técnicas, llamadas **reglas de inferencia**, para extraer consecuencias a partir de un conjunto de enunciados.

Para entender mejor el principio de resolución, vamos a acordar primero representar los enunciados simples mediante letras individuales e indicar la negación de un enunciado mediante el símbolo ¬. Por ejemplo, podemos representar el enunciado “La rana Gustavo es un príncipe” mediante la letra *A* y “La cerdita Peggy es una actriz” mediante la letra *B*. Entonces, la expresión

A OR B

significaría que “La rana Gustavo es un príncipe o la cerdita Peggy es una actriz” y

B AND $\neg A$

significaría que “La cerdita Peggy es una actriz y la rana Gustavo no es príncipe”. Utilizaremos una flecha para indicar que un enunciado “implica” a otro. Por ejemplo, la expresión

$A \rightarrow B$

significa que “La rana Gustavo es un príncipe implica que la cerdita Peggy es una actriz”.

En su forma general, el principio de resolución establece que a partir de dos enunciados de la forma

P OR Q

y

R OR $\neg Q$

podemos concluir el enunciado

P OR R

En este caso decimos que los enunciados originales se resuelven para formar un tercer enunciado, que denominamos **resolvente**. Es importante observar que el resolvente es una consecuencia lógica de los enunciados originales. Es decir, si los enunciados originales son verdaderos, el resolvente también debe serlo. (Si Q es verdadero, entonces R debe ser verdadero; pero si Q es falso, entonces P debe ser verdadero. Por tanto, independientemente de la verdad o falsedad de Q, o P o R deben ser verdaderos.)

Representaremos la resolución de dos enunciados de forma gráfica tal como se muestra en la Figura 6.24, en la que escribimos los enunciados originales con líneas que se proyectan hacia abajo, hasta su resolvente. Observe que el principio de resolución solo se puede aplicar a parejas de enunciados que aparezcan en **forma de cláusula**; es decir, enunciados cuyos componentes elementales estén conectados mediante la operación booleana OR. Así,

P OR Q

está en forma de cláusula, mientras que

$P \rightarrow Q$

no lo está. El hecho de que este problema potencial no plantee ningún obstáculo es consecuencia de un teorema de la lógica matemática que establece que cualquier enunciado expresado en la lógica de predicados de primer orden (un sistema para representar enunciados con una alta potencia expresiva) puede expresarse en forma de cláusula. No vamos a analizar aquí más en detalle este importante teorema, pero observemos, para referencia futura, que el enunciado

$P \rightarrow Q$

es equivalente al enunciado en forma de cláusula

Q OR $\neg P$

Decimos que una colección de enunciados es **inconsistente** si es imposible que todos los enunciados sean ciertos al mismo tiempo. En otras palabras, un conjunto incoherente de enunciados es un conjunto de enunciados que son contradictorios. Un ejemplo simple sería un conjunto que contuviera el enunciado P junto con el enunciado $\neg P$. Los expertos en lógica matemática han demostrado que la aplicación repetida del principio de resolución proporciona un método matemático para confirmar la incoherencia de un conjunto de cláusulas incoherentes. La regla es que si la aplicación repetida del principio de resolución produce la cláusula vacía (el resultado de resolver una cláusula de la forma P con una cláusula de la forma $\neg P$), entonces el conjunto original de enunciados debe ser incoherente. Por ejemplo, en la Figura 6.25 se demuestra que el conjunto de enunciados

P OR Q $\quad$ R OR $\neg Q$ $\quad$ $\neg R$ $\quad$ $\neg P$

es incoherente.

Suponga ahora que queremos confirmar que un conjunto de enunciados implica el enunciado P. Implicar el enunciado P es lo mismo que contradecir el enunciado $\neg P$. Por tanto, para demostrar que el conjunto original de enunciados implica P, lo que tenemos que hacer es aplicar el principio de resolución a los enunciados originales junto con el enunciado $\neg P$ hasta que aparezca una cláusula vacía. Al obtener una cláusula vacía podemos concluir que el enun-

Figura 6.24 Resolución de los enunciados (P OR ¬ Q) y (R OR Q) para generar (P OR R).

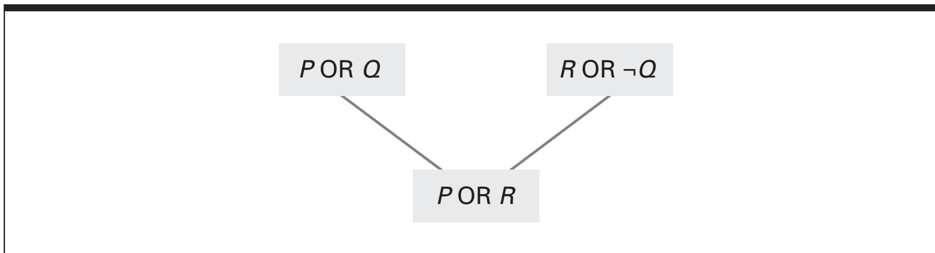
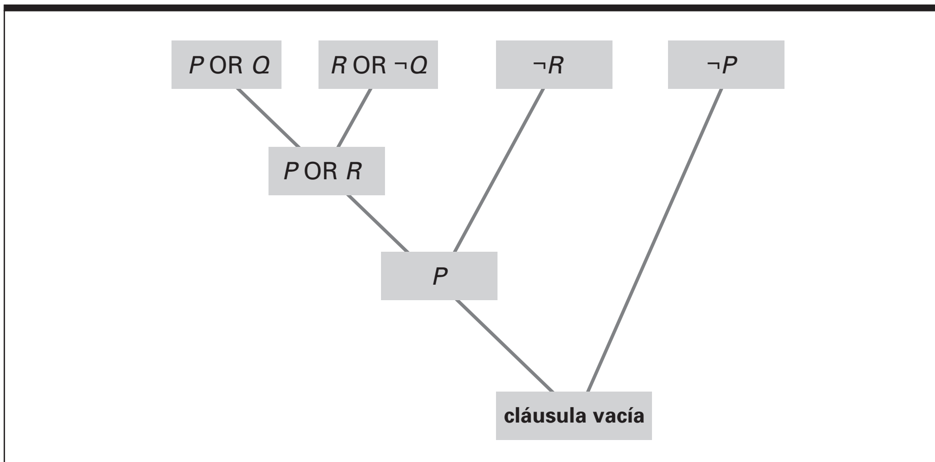


Figura 6.25 Resolución de los enunciados (P OR Q), (R OR ¬ Q), ¬ R y ¬ P.



ciado ¬ P es incoherente con los enunciados originales, por lo que los enunciados originales deben implicar P.

Nos queda por tratar un último punto antes de aplicar el principio de resolución en un entorno de programación real. Suponga que tenemos los dos enunciados siguientes

```
(María está en X) → (El cordero de María está en X)
```

(donde X representa cualquier ubicación) y

```
María está en casa
```

En forma de cláusula, los dos enunciados serían

```
(El cordero de María está en X) OR ¬(María está en X)
```

y

```
(María está en casa)
```

que a primera vista no tienen componentes que puedan ser resueltos. Por otro lado, los componentes `(María está en casa)` y `¬(María está en X)` son casi opuestos el uno al otro. El problema está en darse cuenta de que `María está en X`, siendo un enunciado acerca de cualquier ubicación en general, es también un enunciado acerca de la ***casa*** en particular. Por tanto, un caso especial del primero de los enunciados anteriores sería

```
(El cordero de María está en casa) OR ¬(María está en casa)
```

que puede resolverse con el enunciado

```
(María está en casa)
```

para producir el enunciado

```
(El cordero de María está en casa)
```

El proceso de asignar valores a las variables (como por ejemplo asignar el valor *casa* a *X*) para poder llevar a cabo la resolución se denomina **unificación**. Es este proceso el que permite aplicar enunciados generales a casos específicos en un sistema de deducción lógica.

Prolog

El lenguaje de programación Prolog (abreviatura de PROgramación LOGica) es un lenguaje de programación declarativo cuyo algoritmo subyacente de resolución de problemas se basa en la aplicación repetida del algoritmo de resolución. Dichos lenguajes se denominan lenguajes de **programación lógica**. Un programa en Prolog está compuesto por un conjunto de enunciados iniciales a los que el algoritmo subyacente aplica su razonamiento deductivo. Los componentes a partir de los que se construyen los enunciados se denominan **predicados**. Un predicado está compuesto por un identificador de predicado seguido por una sentencia entre paréntesis que enumera los argumentos de ese predicado. Un único predicado representa un hecho acerca de sus argumentos, y su identificador suele elegirse para reflejar esta semántica subyacente. Así, si queremos expresar el hecho de que Benito es el padre de Marta, podemos utilizar un predicado de la forma

```
padre(benito, marta)
```

Observe que los argumentos de este predicado comienzan por minúscula, aunque representen nombres propios. Esto se debe a que Prolog distingue los argumentos que son constantes de aquellos otros que son variables, insistiendo en que las constantes comiencen por letras minúsculas y las variables con letras mayúsculas. (Aquí hemos utilizado la terminología de la cultura prolog en la que se utiliza el término ***constante*** en lugar del término más genérico ***literal***. Para ser más precisos, el término `benito` [observe la minúscula] se utiliza en prolog para representar el literal que podría representarse como "`Benito`" en una notación más genérica. El término `Benito` [observe la mayúscula] se utiliza en prolog para hacer referencia a una variable.)

Los enunciados en un programa Prolog son o bien hechos o bien reglas, utilizándose un punto para terminar todos ellos. Un hecho está compuesto por un único predicado. Por ejemplo, el hecho de que una tortuga sea más rápida que un caracol podría representarse mediante el enunciado Prolog

```
masrapido(tortuga, caracol).
```

y el hecho de que un conejo sea más rápido que una tortuga podría representarse mediante

```
masrapido(conejo, tortuga).
```

Una regla Prolog es un enunciado de “implicación”. Sin embargo, en lugar de escribir estos enunciados en la forma X → Y, un programador de Prolog escribiría “Y si X”, excepto porque en lugar de la palabra ***si*** se utiliza el símbolo :- (dos puntos seguidos de un guión). Por tanto, la regla “X es viejo implica que X es sabio” podría ser expresada por un experto en lógica de la forma siguiente

```
viejo(X) → sabio(X)
```

mientras que en Prolog se expresa como

```
sabio(X) :- viejo(X).
```

Veamos otro ejemplo. La regla

```
(masrapido(X, Y) AND masrapido(Y, Z)) → masrapido(X, Z)
```

se expresaría en Prolog como sigue

```
masrapido(X, Z) :- masrapido(X, Y), masrapido(Y, Z).
```

(La coma que separa `masrapido(X, Y)` y `masrapido(Y, Z)` representa la conjunción AND.) Aunque reglas como esta no están en forma de cláusula, se permite utilizarlas en Prolog porque pueden convertirse a forma de cláusula fácilmente.

Recuerde que el sistema Prolog no conoce el significado de los predicados de un programa, se limita a manipular los enunciados de una forma completamente simbólica, de acuerdo con la regla de inferencia dictada por el principio de resolución. Por tanto, es responsabilidad del programador describir todas las características pertinentes de un predicado en términos de hechos y reglas. Desde este punto de vista, los hechos tienden a utilizarse en Prolog para casos específicos de un predicado, mientras que las reglas se emplean para describir principios generales. Esta es la técnica seguida en los enunciados anteriores, por lo que respecta al predicado `masrapido`. Los dos hechos describen casos particulares de “mayor rapidez”, mientras que la regla describe una propiedad general. Observe que el hecho de que un conejo sea más rápido que un caracol, aunque se ha enunciado de manera explícita es una consecuencia de combinar los dos hechos indicados con la regla que se ha enunciado.

Al desarrollar software con Prolog, la tarea de un programador es desarrollar el conjunto de hechos y reglas que describen la información que se conoce. Estos hechos y reglas constituyen el conjunto de enunciados iniciales que se emplearán en el sistema deductivo. Una vez establecido este conjunto de enunciados, pueden proponerse al sistema conjeturas (denominadas objetivos en la terminología Prolog), normalmente escribiéndolas en el teclado de una computadora. Cuando se presenta a un sistema Prolog uno de esos objetivos, el sistema aplica el principio de resolución para tratar de confirmar que el objetivo es una consecuencia de los enunciados iniciales. Basándonos en nuestro conjunto de enunciados que describe la relación `masrapido`, cada uno de los objetivos

```
masrapido(tortuga, caracol).
masrapido(conejo, tortuga).
masrapido(conejo, caracol).
```

podría confirmarse de ese modo, porque todos ellos son una consecuencia lógica de los enunciados iniciales. Los dos primeros son idénticos a hechos que

aparecen en el conjunto de enunciados iniciales, mientras que el tercero requiere un cierto grado de deducción por parte del sistema.

Podemos obtener ejemplos más interesantes si proporcionamos objetivos cuyos argumentos sean variables en lugar de constantes. En estos casos, Prolog trata de deducir el objetivo a partir de los enunciados iniciales, al mismo tiempo que observa las unificaciones requeridas para hacerlo. Entonces, si se obtiene el objetivo, Prolog informa acerca de cuáles han sido esas unificaciones. Por ejemplo, considere el objetivo

```
masrapido(W, caracol).
```

En respuesta a esto, Prolog respondería

```
masrapido(tortuga, caracol).
```

De hecho, esta es una consecuencia de los enunciados iniciales y concuerda con el objetivo mediante la aplicación de la regla de unificación. Además, si pidiéramos a Prolog que nos diera más información, el sistema encontraría la consecuencia

```
masrapido(conejo, caracol).
```

Por el contrario, podemos pedir a Prolog que halle casos de animales que sean más lentos que un conejo proponiéndole el objetivo

```
masrapido(conejo, W).
```

De hecho, si partiéramos del objetivo

```
masrapido(V, W).
```

Prolog buscaría todas las relaciones que puedan derivarse de los enunciados iniciales y que indiquen que un animal es más rápido que otro. Esto indica que podría emplearse un único programa Prolog por ejemplo para confirmar que un animal concreto es más rápido que otro, para encontrar aquellos animales que son más rápidos que un animal dado, para encontrar aquellos animales que sean más lentos que un animal determinado, o para encontrar todas las relaciones que indiquen que un animal es más rápido que otro.

Esta versatilidad potencial es una de las características de Prolog que ha atraído a los expertos de las Ciencias de la computación. Lamentablemente, implementado en un sistema Prolog, el procedimiento de resolución hereda una serie de limitaciones que no están presentes en su forma teórica, por lo que los programas Prolog podrían no llegar a satisfacer esa flexibilidad prevista. Para entender qué es lo que queremos decir, observe primero que el diagrama de la Figura 6.25 muestra únicamente aquellas resoluciones que son pertinentes para la tarea que tenemos entre manos. Existen otras direcciones que el proceso de resolución podría seguir. Por ejemplo, las cláusulas de más a la izquierda y de más a la derecha se podrían resolver para generar el resolvente Q. Así, además de los enunciados que describen los hechos y reglas básicos relativos a una aplicación, un programa Prolog debe contener a menudo enunciados adicionales cuyo propósito sea el de guiar el proceso de resolución en la dirección correcta. Por esta razón, los programas Prolog reales pueden no llegar a plasmar la multiplicidad de objetivos sugerida por nuestro anterior ejemplo.

Cuestiones y ejercicios

1. ¿Cuáles de los enunciados *R, S, T, U* y *V* son consecuencia lógica del conjunto de enunciados (¬*R* OR *T* OR *S*), (¬*S* OR *V*), (¬*V* OR *R*), (*U* OR ¬*S*), (*T* OR *U*) y (*S* OR *V*)?
2. ¿Es coherente el siguiente conjunto de enunciados? Explique su respuesta.

   P OR *Q* OR *R* ¬*R* OR *Q* *R* OR ¬*P* ¬*Q*

3. Complete las dos reglas situadas al final del programa Prolog que se muestra a continuación, de modo que el predicado `madre(X, Y)` signifique "X es la madre de Y" y el predicado `padre(X, Y)` signifique "X es el padre de Y."

```
hembra(carolina).
hembra(susana).
varón(benito).
varón(juan).
progenitor(juan, carolina).
progenitor(susana, carolina).
madre(X,Y) :-
padre(X,Y) :-
```

4. En el contexto del programa Prolog de la Cuestión 3, la siguiente regla pretende significar que `X` es hermano de `Y` si `X` e `Y` tienen un progenitor en común.

```
hermano(X, Y) :- progenitor(Z, X), progenitor(Z, Y).
```

   ¿Qué conclusión inesperada podría realizar Prolog si utilizamos esta definición de la relación entre hermanos?

Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. ¿Cuales son las desventajas de utilizar un lenguaje ensamblador?
2. Traduzca el siguiente programa de pseudocódigo al lenguaje máquina descrito en el Apéndice C.

```
x ← 0;
while (x < 3) do
  (x ← x + 1)
```

3. Traduzca la sentencia

```
Mitad ← Longitud + Ancho
```

   al lenguaje máquina del Apéndice C, suponiendo que `Longitud`, `Ancho` y `Mitad` están todos ellos representados en notación de punto flotante.

4. Traduzca la sentencia de alto nivel

```
if (X igual a 0)
  then Z ← Y + W
  else Z ← Y + X
```

   al lenguaje máquina del Apéndice C, suponiendo que `W`, `X`, `Y` y `Z` son todos ellos valores representados en notación de complemento a dos, utilizando cada uno un byte de memoria.

5. ¿Cuáles son las principales características de la programación orientada a aobjetos?
6. ¿Cuáles son los diferentes tipos de error posibles en un programa?
7. Suponga que tenemos dos funciones *f* y *g*, que esperan que se les entreguen dos valores numéricos como entrada. La función *f* devuelve el mayor de esos dos valores y la función *g* devuelve el menor de dichos dos valores de entrada. Si *w*, *x*, *y* y *z* representan valores numéricos, ¿cuál será el resultado devuelto por *g(f(w,x), g(y,z))*?
8. Suponga que *f* es una función que devuelve el resultado de eleiminar los caracteres pares de la cadena de símbolos que se le proporciona como entrada y que *g* es una función que devuelve el resultado de eliminar los caracteres impares de la cadena dada como entrada. Si *x* es la cadena "*abcdefghijk*" y el índice inicial de "a" es 1, ¿qué es lo que devolverá *g(f(x))*?
9. Suponga que vamos a escribir un programa orientado a objetos para mantener nuestros registros financieros. ¿Qué datos habría que almacenar dentro del objeto que representa nuestra cuenta corriente? ¿A qué mensajes debería ser capaz de responder dicho objeto? ¿Qué otros objetos podrían utilizarse en el programa?
10. Resuma la diferencia entre un lenguaje de alto nivel y un lenguaje ensamblador.
11. Diseñe un lenguaje ensamblador para la máquina descrita en el Apéndice C.
12. Un programador argumenta que la capacidad de declarar constantes en un programa no es necesaria porque se pueden usar variables en su lugar. Por ejemplo, el ejemplo de la altitud del aeropuerto (`AltAeropuerto`) de la Sección 6.2 podría resolverse declarando `AltAeropuerto` como una variable y luego asignándole el valor requerido al principio del programa. ¿Por qué esta solución no es tan adecuada como la de emplear una constante?
13. Resuma la diferencia entre sentencias declarativas y sentencias imperativas.
14. Explique la diferencia entre literal, constante y variable.
15. a. ¿Qué es la precedencia de operadores?
    b. Dependiendo de la precedencia de los operadores, ¿qué valores podrían asociarse con la expresión $6 + 2 \times 3$?
16. ¿Qué es la programación estructurada?
17. ¿Cuál es la diferencia entre el significado del símbolo de "igualdad" en la sentencia

```
if (X = 5) then ( . . . )
```

    y en la sentencia de asignación

```
X = 2 + Y
```

18. Dibuje un diagrama de flujo que represente la estructura expresada por la siguiente sentencia `for`

```
for (int x = 10; x > 1; --x)
{ . . . }
```

19. Traduzca la siguiente sentencia `for` a un segmento de programa equivalente que utilice la sentencia `while` en nuestro pseudocódigo del Capítulo 5.

```
for (int x = 15; x > 3; --x)
{ . . . }
```

20. Si está familiarizado con la escritura de música, analice la notación musical como un lenguaje de programación. ¿Cuáles son las estructuras de control? ¿Cuál es la sintaxis para insertar comentarios en el programa? ¿Qué notación musical tiene una semántica similar a la de la sentencia `for` de la Figura 6.7?
21. Dibuje un diagrama de flujo que represente la estructura expresada por la siguiente sentencia.

```
x =8
if (y < 7) then
y = y +3
   if (y< 7) then
   z = z - 6
   endif
   elseif (x < 5) then
y = y -8
endif
```

22. Resuma la siguiente rutina confusa utilizando una única sentencia `if-then-else`.

```
    if X > 15 then goto 60
    X = X-9
    goto 110
60  X = X *7
110 stop
```

23. Resuma la siguiente rutina confusa utilizando una única sentencia `if-then-else`:

```
    if X > 5 then goto 60
    X = X - 9
    goto 110
60  X = X * 7
110 stop
```

24. Resuma las estructuras de control básicas que podemos encontrar en los lenguajes de programación imperativos y orientados a objetos para la realización de cada una de las siguientes actividades:

a. Determinar qué comando debe ejecutarse a continuación.

b. Repetir un conjunto de comandos.

c. Modificar el valor de una variable.

25. ¿Qué es el software multiplataforma?

26. Suponga que declaramos que la variable `X` en un programa es de tipo entero. ¿Qué error se produciría al ejecutar la sentencia de programa

```
X ← 'A'
```

27. ¿Qué es un lenguaje de formato libre?

28. ¿Por qué probablemente no pasaríamos por valor a un procedimiento una matriz de gran tamaño?

29. Suponga que definimos el procedimiento `Modificar` en nuestro pseudocódigo del Capítulo 5 mediante

```
procedure Modificar (Y)
 Y ← 7;
 imprimir el valor de Y.
```

Si pasamos los parámetros por valor, ¿qué se imprimiría al ejecutar el siguiente segmento de código? ¿Y si pasáramos los parámetros por referencia?

```
X ← 5;
aplicar procedimiento Modificar a X;
imprimir el valor de X;
```

30. Suponga que definimos el procedimiento `Modificar` en nuestro pseudocódigo del Capítulo 5 mediante

```
procedure Modificar (Y)
 Y ← 9;
 imprimir el valor de X;
 imprimir el valor de Y.
```

Suponga también que X es una variable global. Si se pasan los parámetros por valor, ¿que se imprimiría al ejecutar el siguiente segmento de programa? ¿Y si pasáramos los parámetros por referencia?

```
X ← 5;
aplicar procedimiento Modificar a X;
imprimir el valor de X;
```

31. En ocasiones un argumento se pasa a un procedimiento generando un duplicado para que el procedimiento lo utilice (al igual que cuando el parámetro se pasa por valor), pero cuando el procedimiento se completa el valor contenido en la copia del procedimiento se transfiere al parámetro real, antes de que el procedimiento llamante continúe. En estos casos, decimos que el parámetro se pasa por valor-resultado. ¿Qué imprimiría el fragmento de código de programa del problema 30 si los parámetros se pasaran por valor-resultado?

32. a. ¿Cuál es la ventaja de pasar los parámetros por valor en lugar de pasarlos por referencia?

b. ¿Cuál es la ventaja de pasar los parámetros por referencia en lugar de pasarlos por valor?

33. ¿Qué ambigüedad existe en la siguiente sentencia?

```
X ← 3 + 2 × 5
```

34. Suponga que una pequeña empresa tiene cinco empleados y pretende incrementar dicho número a seis. Además, suponga que uno de los programas de la empresa contiene las siguientes sentencias de asignación.

```
SalarioDiario = TotalSal/5;
SalarioMedio = TotalSal/5;
VentasDiarias = TotalVentas/5;
VentasMedias = TotalVentas/5;
```

¿Cómo podría simplificarse la tarea de actualizar el programa si este se hubiera escrito originalmente utilizando dos constantes denominadas `NumEmpl` y `DiasSem` (y a ambas se les hubiera asignado el valor 5) de modo que las sentencias de asignación pudieran expresarse como

```
SalarioDiario = TotalSal/DiasSem;
SalarioMedio = TotalSal/NumEmpl;
VentasDiarias = TotalVentas/DiasSem;
VentasMedias = TotalVentas/NumEmpl;
```

35. a. ¿Cuál es la diferencia entre un lenguaje formal y un lenguaje natural?
b. Proporcione un ejemplo de cada uno de ellos.

36. Dibuje un diagrama sintáctico que represente la estructura de la sentencia `for` correspondiente al pseudocódigo del Capítulo 5.

37. Diseñe un conjunto de diagramas sintácticos para describir la sintaxis de los números telefónicos utilizados en su país. Por ejemplo, en Estados Unidos, los números de teléfono están formados por un código de área, seguido de un código regional, seguido de un número de cuatro dígitos, como por ejemplo (444) 555–1234.

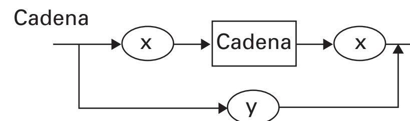
38. Explique por qué el uso de sentencias `goto` hace que un lenguajes sea desectructurado.

39. Diseñe un conjunto de diagramas sintácticos para describir diferentes formas de representar fechas, como por ejemplo *mes/día/año* o *mes día, año*.

40. Diseñe un conjunto de diagramas sintácticos que describa la estructura gramatical de las "frases" compuestas por apariciones de la palabra *sí* seguidas por el mismo número de la palabra *no*. Por ejemplo, " sí sí no no" sería una de dichas frases, mientras que "no sí", "sí no no" y "sí no sí" no serían frases permitidas.

41. Proporcione un argumento para sostener que no puede diseñarse un conjunto de diagramas sintácticos que describa la estructura gramatical de las "frases" compuestas por apariciones de la palabra *sí*, seguidas del mismo número de apariciones de la palabra *no*, seguidas por el mismo número de apariciones de la palabra *quizá*. Por ejemplo, "sí no quizá" y "sí sí no no quizá quizá" serían frases permitidas, mientras que "sí quizá", "sí no no quizá quizá" y "quizá no" no serían frases permitidas.

42. Escriba una frase que describa la estructura de la cadena xxaaaddykkbbbxx.



43. Añada diagramas sintácticos a los de la Cuestión 5 de la Sección 6.4 para obtener un conjunto de diagramas que definan la estructura Danza como un Chachacha o un Vals, estando un Vals compuesto por una o más copias del patrón

adelante diagonal juntar

o

atrás diagonal juntar

44. Dibuje el árbol sintáctico de la expresión

$x \times y + y \div x$

basándose en los diagramas sintácticos de la Figura 6.15.

45. ¿Qué optimización de código podría realizar un generador de código a la hora de construir el código máquina que represente la siguiente sentencia?

```
if (X = 5) then (Z ← X + 2)
           else (Z ← X + 4)
```

46. Simplifique el siguiente fragmento de código de programa

```
Y ← 5;
if (Y = 7)
  then (Z ← 8)
  else (Z ← 9)
```

47. Simplifique el siguiente fragmento de código de programa

```
X ← 9;
while (X igual a 10) do
  (X ← 5)
X ← 9;
```

48. ¿Cuál es la diferencias entre una función y un procedimiento en el lenguaje de programación C?

49. Indique cómo puede utilizarse la herencia para desarrollar clases que describan distintos tipos de edificios.

50. En la programacion orientada a objetos, ¿qué es la encapsulación?

51. a. Proporcione un ejemplo de una situación en la que una variable de instancia debería ser privada.

b. Proporcione un ejemplo de una situación en la que una variable de instancia debería ser pública.

c. Proporcione un ejemplo de una situación en la que un método debería ser privado.

d. Proporcione un ejemplo de una situación en la que un método debería ser público.

52. Describa algunos objetos que podríamos encontrar en un programa utilizado para simular el tráfico de peatones en la recepción de un hotel. Incluya explicaciones de las acciones que algunos de esos objetos deberían poder realizar.

***53.** ¿Qué significa el término "monitor" en el contexto de un lenguaje de programación?

***54.** ¿Qué propiedades del procesamiento concurrente hacen deseable utilizar un lenguaje de programación que permita el uso de la concurrencia?

***55.** Dibuje un diagrama (similar al de la Figura 6.25) que represente las resoluciones necesarias para demostrar que el conjunto de enunciados (Q OR $\neg R$), (T OR R), $\neg P$, (P OR $\neg T$) y (P OR $\neg Q$) son inconsistentes.

***56.** Escriba un programa en Prolog para calcular el factorial de un número dado.

57.** Amplíe el programa Prolog esbozado en las Cuestiones 3 y 4 de la Sección 6.7 para incluir las relaciones familiares adicionales ***tío, ***tía***, ***abuelo*** y ***primo***. Añada también una regla que defina `padres (X, Y, Z)` para indicar que `X` e `Y` son los padres de `Z`.

***58.** Suponiendo que la primera sentencia del siguiente programa Prolog pretende significar que "A Alicia le gustan los deportes", traduzca los dos últimos enunciados del programa. A continuación, enumere todas las cosas que Prolog, basándose en este programa, podría deducir que le gustan a Alicia. Explique su respuesta.

```
gusta(alicia, deportes).
gusta(alicia, musica).
gusta(carolina, musica).
gusta(david, X) :- gusta(X, deportes).
gusta(alicia, X) :- gusta(david, X).
```

***59.** ¿Qué problema nos encontraríamos si se ejecutara el siguiente fragmento de código de programa en una computadora en la que los valores se representaran en el formato de punto flotante de ocho bits descrito en la Sección 1.7?

```
X ← 0.01;
while (X distinto de 1.00) do
  (imprimir el valor de X;
   X ← X + 0.01)
```

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. En general, las leyes de propiedad intelectual protegen los derechos de propiedad asociados con la expresión de una idea, pero no protegen la propia idea. Como resultado, un párrafo de un libro está sujeto a derechos de propiedad, pero las ideas expresadas en el párrafo no lo están, ¿cómo debería extenderse este derecho al código fuente y a los algoritmos que expresan? ¿Hasta qué punto debería permitirse que una persona que conoce los algoritmos utilizados en un paquete software comercial escribiera su propio programa expresando esos mismos algoritmos y comercializara esa nueva versión del software?
2. Utilizando un lenguaje de programación de alto nivel, un programador puede expresar algoritmos utilizando palabras tales como ***if***, ***then*** y ***while***. ¿Hasta qué punto comprende la computadora esas palabras? ¿La capacidad de responder correctamente al uso de palabras implica que se comprenden esas palabras? ¿Cómo sabe usted cuándo otra persona ha comprendido lo que ha dicho?
3. ¿Debería tener derecho a beneficiarse del uso de un lenguaje de programación la persona que ha desarrollado un lenguaje de programación nuevo y de gran utilidad? En caso afirmativo, ¿cómo podría protegerse ese derecho? ¿Hasta qué punto puede ser alguien propietario de un lenguaje? ¿Hasta qué punto debería una empresa tener derecho de propiedad sobre los logros creativos e intelectuales de sus empleados?
4. Cuando se está aproximando la fecha de entrega, ¿es aceptable que un programador deje de documentar un programa introduciendo sentencias de comentario, con el fin de conseguir que el programa funcione a tiempo? (Los estudiantes novatos suelen sorprenderse al ver lo importante que se considera la documentación entre los desarrollados de software profesionales.)
5. Buena parte de la investigación relativa a los lenguajes de programación ha consistido en el desarrollo de lenguajes que permitan a los programadores escribir programas que puedan ser fácilmente leídos y comprendidos por los seres humanos. ¿Hasta qué punto debería exigirse a un programador que utilice esas capacidades? Es decir, ¿hasta qué punto es suficiente que el programa funcione correctamente, aunque no esté bien escrito desde la perspectiva de quien tenga que leerlo?
6. Suponga que un programador principiante escribe un programa para su propio uso y al hacerlo, es perezoso en lo que respecta a la adecuada construcción del programa. El programa no utiliza las características del lenguaje de programación que permitirían hacerlo más legible, no es tampoco eficiente y contiene una serie de atajos que se aprovechan de la situación

concreta en la que el programador pretende utilizar el programa. Con el tiempo, el programador proporciona copias del programa a amigos suyos que también quieren usarlo y estos amigos se lo proporcionan a otros amigos. ¿Hasta qué punto sería responsable el programador de los problemas que pudieran surgir?

7. ¿Hasta qué punto debería un profesional del campo de la computación conocer los distintos paradigmas de programación? Algunas empresas insisten en que todo el software desarrollado dentro de la empresa se escriba utilizando el mismo lenguaje de programación predeterminado. ¿Cambiaría su respuesta a la pregunta original si nuestro profesional de la programación trabajara para una de esas empresas?

Lecturas adicionales

Aho, A. V., M. S. Lam, R. Sethi y J. D. Ullman. ***Compilers: Principles, Techniques, and Tools***, 2ª ed. Boston, MA: Addison-Wesley, 2007.

Barnes, J. ***Programming in Ada 2005***. Boston, MA: Addison-Wesley, 2006.

Clocksin, W. F. y C. S. Mellish. ***Programming in Prolog,*** 5ª ed. Nueva York: Springer-Verlag, 2003.

Friedman, D. P. y M. Felleisen. ***The Little Schemer,*** 4ª ed. Cambridge, MA: MIT Press, 1995.

Hamburger, H. y D. Richards. ***Logic and Language Models for Computer Science***. Upper Saddle River, NJ: Prentice-Hall, 2002.

Kernighan, B. W. y D. M. Ritchie. ***The C Programming Language***, 2ª ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

Metcalf, M. y J. Reid. ***Fortran 90/95 Explained,*** 2ª ed. Oxford, England: Oxford University Press, 1999.

Pratt, T. W. y M. V. Zelkowitz. ***Programming Languages, Design and Implementation,*** 4ª ed. Upper Saddle River, NJ: Prentice-Hall, 2001.

Savitch, W. ***Absolute C++,*** 3ª ed. Boston, MA: Addison-Wesley, 2008.

Savitch, W. ***Absolute Java,*** 3ª ed. Boston, MA: Addison-Wesley, 2008.

Savitch, W. ***Problem Solving with C++,*** 6ª ed. Boston, MA: Addison-Wesley, 2008.

Scott, M. L. ***Programming Language Pragmatics,*** 3ª ed. Nueva York: Morgan Kaufmann, 2009.

Sebesta, R. W. ***Concepts of Programming Languages,*** 9ª ed. Boston, MA: Addison-Wesley, 2009.

Wu, C. T. ***An Introduction to Object-Oriented Programming with Java***, 3ª ed. Burr Ridge, IL: McGraw-Hill, 2008.

CAPÍTULO

7

Ingeniería del software

En este capítulo vamos a explorar los problemas que podemos encontrarnos durante el desarrollo de sistemas software complejos y de gran tamaño. Este tema se denomina *ingeniería del software* porque el desarrollo de software es un proceso de ingeniería. El objetivo de los investigadores en el campo de la ingeniería del software es encontrar principios que sirvan como guía en el proceso del desarrollo del software y que conduzcan a la obtención de productos software eficientes y fiables.

La ingeniería del software es la rama de las Ciencias de la computación que busca principios que sirvan como guía para el desarrollo de sistemas software complejos y de gran tamaño. Los problemas con los que nos encontramos al desarrollar este tipo de sistemas son algo más que versiones ampliadas de los problemas con los que topamos al escribir pequeños programas. Por ejemplo, el desarrollo de esos sistemas requiere el esfuerzo de más de una persona a lo largo de un periodo dilatado de tiempo durante el cual los requisitos del sistema propuesto pueden verse alterados y el personal asignado al proyecto puede variar. En consecuencia, la ingeniería del software incluye temas tales como la gestión de personal y de proyectos, que suelen asociarse más con la gestión empresarial que con las Ciencias de la computación. Nosotros, sin embargo, nos centraremos en los temas más estrechamente relacionados con las Ciencias de la computación.

7.1 La disciplina de la ingeniería del software

Para poder apreciar los problemas implicados en la ingeniería del software resulta útil seleccionar un dispositivo complejo de gran tamaño (un automóvil, un edificio de oficinas de varios pisos, o quizá una catedral) e imaginar que nos piden que lo diseñemos y que luego supervisemos su construcción. ¿Cómo podemos estimar el coste en tiempo, dinero y otros recursos necesarios para completar el proyecto? ¿Cómo podemos dividir el proyecto en piezas más manejables? ¿Cómo podemos garantizar que las piezas producidas sean compatibles? ¿Cómo pueden comunicarse las personas que estén trabajando en las distintas piezas? ¿Cómo podemos medir el progreso? ¿Cómo podemos manejar el amplio rango de detalles (la selección de los pomos de las puertas, el diseño de las gárgolas, la disponibilidad de cristal de color azul para las vidrieras, la fortaleza de los pilares, el diseño de los conductos para el sistema de calefacción)? Durante el desarrollo de un sistema software de gran envergadura se hace necesario responder a cuestiones del mismo tipo.

Puesto que la ingeniería es un campo bien consolidado, podríamos pensar que existe una gran cantidad de técnicas de ingeniería previamente desarrolladas que puedan ser útiles para responder a las cuestiones anteriores. Esta forma de razonar es parcialmente cierta, pero olvida algunas diferencias fundamentales existentes entre las propiedades del software y las de otros campos de la ingeniería. Estas diferencias han planteado numerosos problemas a los proyectos de ingeniería del software, lo que ha terminado conduciendo a que los costes se sobrepasaran, a que los productos se entregaran con retraso y a que los clientes estuvieran insatisfechos. A su vez, la identificación de estas diferencias ha resultado ser el primer paso a la hora de hacer avanzar la disciplina de la ingeniería del software.

Una de esas diferencias implica la capacidad de construir sistemas a partir de componentes genéricos prefabricados. Los campos tradicionales de la ingeniería se han aprovechado desde hace tiempo de la capacidad de utilizar componentes disponibles comercialmente como bloques componentes, a la hora de construir dispositivos complejos. El diseñador de un nuevo automóvil no tiene que diseñar un nuevo motor o un sistema de transmisión, sino que utiliza versiones anteriormente diseñadas de dichos componentes. Sin embargo, la inge-

niería del software está retrasada en este aspecto. En el pasado, los componentes software previamente diseñados eran específicos de cada dominio; es decir, su diseño interno estaba basado en una aplicación específica, y por tanto su uso como componentes genéricos estaba limitado. El resultado es que los sistemas software complejos se han construido tradicionalmente partiendo de cero. Como veremos en este capítulo, se están haciendo progresos significativos a este respecto, aunque todavía queda mucho trabajo por hacer.

Otra diferencia entre la ingeniería del software y otras disciplinas de la ingeniería es la falta de técnicas cuantitativas, denominadas **métricas**, para medir las propiedades del software. Por ejemplo, para proyectar el coste de desarrollar un sistema software, a uno le gustaría estimar la complejidad del producto propuesto, pero los métodos para medir la "complejidad" del software son ellos mismos muy difusos. De forma similar, la evaluación de la calidad de un producto software resulta muy complicada. En el caso de los dispositivos mecánicos, una importante medida de la calidad es el tiempo medio entre fallos, que es esencialmente una medida de lo bien que un dispositivo soporta el desgaste. Por el contrario, el software no se desgasta, así que este método de medir la calidad no es aplicable en la ingeniería del software.

Las dificultades inherentes a la medición de las propiedades del software de una forma cuantitativa es una de las razones de que la ingeniería del software haya tenido que luchar tanto para encontrar una base tan rigurosa como la ingeniería mecánica y eléctrica. Mientras que estos otros campos se basan en la ciencia de la Física muy bien establecida, la ingeniería del software continúa tratando de encontrar sus raíces.

Por ello, la investigación en el campo de la ingeniería del software trata de progresar actualmente en dos niveles: algunos investigadores, los denominados prácticos, trabajan intentando desarrollar técnicas para su aplicación inmediata, mientras que otros, los teóricos, buscan los principios y teorías subyacentes con los que algún día puedan construirse técnicas más estables. Estando fundamentadas en bases subjetivas, muchas metodologías desarrolladas y promovidas por los prácticos en el pasado han sido sustituidas por otras técnicas que podrían ellas mismas llegar a quedar obsoletas con el tiempo. Mientras tanto, los progresos realizados por los teóricos continúan siendo lentos.

La necesidad de progreso tanto de los prácticos como de los teóricos es enorme. Nuestra sociedad se ha hecho adicta a los sistemas de computadoras y su software asociado. Nuestra economía, la salud pública, el gobierno, las fuerzas de seguridad, el transporte y la defensa dependen de sistemas software de gran envergadura. A pesar de lo cual, continúa habiendo graves problemas de fiabilidad en ese tipo de sistemas. Los errores en el software han provocado desastres y casi desastres, tales como que la Luna naciente fuera interpretada como un ataque nuclear, que el Banco de Nueva York perdiera cinco millones de dólares en un solo día, la pérdida de sondas espaciales, la exposición a sobredosis de radiación que han causado muertes o han provocado graves secuelas a personas y la interrupción simultánea de las comunicaciones telefónicas en amplias regiones.

Esto no quiere decir que la situación sea catastrófica. Se están realizando grandes progresos a la hora de resolver problemas tales como la falta de métricas o de componentes prefabricados. Además, la aplicación de tecnología de computadoras al proceso de desarrollo del software, que ha dado como resul-

ACM

La ACM (*Association for Computing Machinery*, Asociación de hardware de computación) fue fundada en 1947 como una organización internacional de carácter científico y educativo dedicada al progreso de la ingeniería, las ciencias y las aplicaciones de la tecnología de la información. Tiene su sede en Nueva York e incluye numerosos grupos especiales de trabajo (SIG, *Special Interest Groups*) que se centran en temas tales como la arquitectura de computadoras, la inteligencia artificial, la computación biomédica, la interacción entre computadoras y sociedad, la pedagogía de las Ciencias de la computación, los gráficos por computadora, hipertexto/hipermedia, sistemas operativos, lenguajes de programación, simulación y modelado e ingeniería del software. El sitio web de la ACM está disponible en la dirección `http://www.acm.org`. Su Código de ética y conducta profesional puede consultarse en `http://www.acm.org/constitution/code.html`.

tado lo que se denomina **ingeniería del software asistida por computadora** (CASE, *Computer-aided software engineering*), continúa simplificando y facilitando el proceso de desarrollo del software. La tecnología CASE ha conducido al desarrollo de diversos sistemas computerizados, conocidos con el nombre de **herramientas CASE**, que incluyen sistemas de planificación de proyectos (como ayuda para la estimación de costes, para la fijación de hitos en los proyectos y para la asignación de personal), sistemas de gestión de proyectos (como ayuda para la monitorización del progreso del proyecto de desarrollo), herramientas de documentación (como ayuda para la escritura y organización de la documentación), sistemas de prototipado y simulación (como ayuda al desarrollo de prototipos), sistemas de diseño de interfaces (como ayuda para el desarrollo de interfaces gráficas de usuario) y sistemas de programación (como ayuda a la escritura y depuración de programas). Algunas de estas herramientas son poco más que los procesadores de textos, el software de hoja de cálculo y los sistemas de comunicación por correo electrónico originalmente desarrollados para uso genérico y adoptados por los ingenieros de software. Otros son paquetes muy sofisticados diseñados específicamente para los entornos de ingeniería del software. De hecho, los sistemas conocidos con el nombre de **entornos integrados de desarrollo** (IDE, *Integrated Development Environments*) combinan herramientas para el desarrollo de software (editores, compiladores, herramientas de depuración, etc.) en un único paquete integrado. Un ejemplo notable de dichos sistemas son los que se utilizan para el desarrollo de aplicaciones de teléfonos inteligentes. Estos sistemas no solo proporcionan las herramientas necesarias para depurar y escribir el software, sino que también proporcionan simuladores que, por medio de pantallas gráficas, permiten a un programador ver cómo se comportará en realidad en un teléfono el software que se está desarrollando.

Además de los esfuerzos de los investigadores, una serie de organizaciones de carácter profesional y de normalización, incluyendo a la ISO, la ACM y el IEEE, se han unido a la batalla de tratar de mejorar el estado de la ingeniería del software. Esta serie de esfuerzos va desde la adopción de códigos éticos de conducta profesional que mejoren la profesionalidad de los desarrolladores de software y les animen a asumir sus responsabilidades individuales hasta el esta-

blecimiento de estándares para medir la calidad de las organizaciones de desarrollo software y proporcionar directrices que ayuden a esas organizaciones a mejorar su manera de trabajar.

En el resto del capítulo hablaremos de algunos de los principios fundamentales de la ingeniería del software (como el ciclo de vida del software y la modularidad), examinaremos algunas de las direcciones en las que la ingeniería del software está progresando (como la identificación y aplicación de patrones de diseño y la aparición de componentes software reutilizables), y analizaremos los efectos que el paradigma de la orientación a objetos ha tenido en este campo.

Cuestiones y ejercicios

1. ¿Por qué el número de líneas de un programa no sería una buena medida de la complejidad del programa?
2. Sugiera una métrica para medir la calidad del software. ¿Qué debilidades tiene su métrica?
3. ¿Qué técnica puede utilizarse para determinar cuántos errores hay en una unidad de software?
4. Identifique dos contextos en los que el campo de la ingeniería software haya estado progresando y mejorando, o lo esté haciendo actualmente.

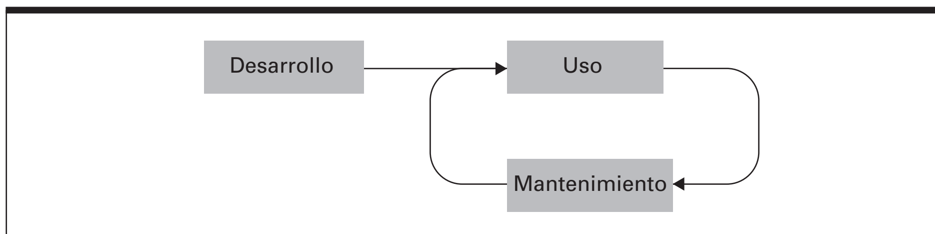
7.2 El ciclo de vida del software

El concepto más fundamental en la ingeniería del software es el ciclo de vida del software.

El ciclo en su conjunto

El ciclo de vida del software se muestra en la Figura 7.1. Esta figura representa el hecho de que una vez que el software ha sido desarrollado, entra en un ciclo de utilización y mantenimiento, un ciclo que continúa durante el resto de la vida útil de ese software. Ese patrón es común también para muchos otros productos manufacturados. La diferencia es que en el caso de otros productos, la fase de mantenimiento tiende a ser un proceso de reparación, mientras que en el caso del software, la fase de mantenimiento suele consistir en correcciones y

Figura 7.1 El ciclo de vida del software.



actualizaciones. De hecho, el software pasa a la fase de mantenimiento debido a que se descubren errores, se producen cambios en la aplicación práctica del software que requieren los cambios correspondientes en el propio software o porque se descubre que los cambios realizados durante una modificación anterior inducen problemas en algún otro lugar del software.

Independientemente de por qué pasa el software a la fase de mantenimiento, el proceso requiere que una persona (que a menudo no es el autor original) estudie el programa subyacente y su documentación hasta entender el programa o al menos la parte pertinente del mismo. De no hacerse así, cualquier modificación podría introducir más problemas de los que resuelve. El llegar a comprender el software puede ser una tarea difícil, aún cuando esté bien diseñado y documentado. De hecho, es a menudo durante esta fase que un determinado software se descarta con la pretensión (que muy a menudo es correcta) de que resulta más fácil desarrollar un nuevo sistema partiendo de cero que modificar adecuadamente el paquete existente.

La experiencia demuestra que un pequeño esfuerzo durante la fase de desarrollo del software puede marcar una enorme diferencia en el momento en que se requieran modificaciones. Por ejemplo, en nuestras explicaciones acerca de la descripción de datos en el Capítulo 6 hemos visto cómo el uso de constantes en lugar de literales puede simplificar enormemente los futuros cambios. A su vez, la mayor parte de los esfuerzos de investigación en el campo de la ingeniería del software se centran en la etapa de desarrollo del ciclo de vida del software, siendo el objetivo aprovechar al máximo la relación esfuerzo-beneficio existente en esta fase.

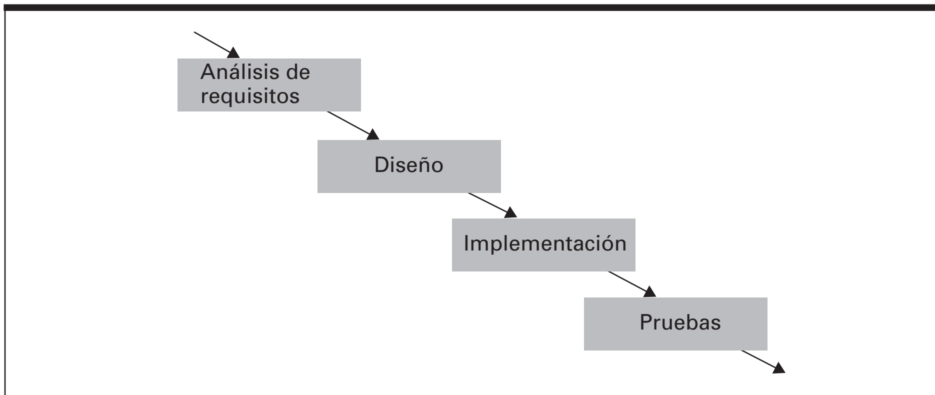
La fase de desarrollo tradicional

Los pasos principales en la fase del desarrollo tradicional del ciclo de vida del software son: el análisis de requisitos, el diseño, la implementación y las pruebas (Figura 7.2).

Análisis de requisitos El ciclo de vida del software comienza con el análisis de requisitos, cuyo objetivo consiste en especificar qué servicios proporcionará el sistema propuesto, identificar las condiciones impuestas a esos servicios (restricciones temporales, de seguridad, etc.) y definir cómo interactuará el mundo exterior con el sistema.

El análisis de requisitos requiere que aporten una gran cantidad de datos **todas las partes interesadas** (los usuarios futuros, así como el resto de las personas que tengan algún vínculo con el proyecto, por ejemplo de naturaleza legal o financiera) en el sistema propuesto. De hecho, en aquellos casos en los que el usuario final es una entidad, como por ejemplo una empresa o un organismo gubernamental, que pretende contratar a un desarrollador de software para llevar a cabo el proyecto software, el análisis de requisitos puede comenzar mediante un estudio de factibilidad realizado exclusivamente por el usuario. En otros casos, el desarrollador del software puede dedicarse al negocio de fabricar software **comercial** para el mercado de masas, quizá para venderlo en tiendas minoristas o para que sea descargado a través de Internet. En este caso, el usuario es una entidad definida con mucha menos precisión y el análisis de requisitos puede comenzar con un estudio de mercado realizado por el propio desarrollador del software.

Figura 7.2 Fase de desarrollo tradicional del ciclo de vida del software.



En cualquier caso, el proceso de análisis de los requisitos consiste en compilar y analizar las necesidades del usuario del software; negociar entre las partes interesadas el proyecto hasta definir una serie de compromisos entre los deseos, las necesidades, los costes y la factibilidad; y finalmente desarrollar un conjunto de requisitos que identifique las características y servicios que el sistema software terminado debe tener. Estos requisitos se registran en un documento denominado **especificación de requisitos software**. En cierto sentido, este documento es un acuerdo escrito entre todas las partes implicadas, que pretende guiar el proceso de desarrollo del software y proporcionar un medio de resolver las disputas que puedan surgir posteriormente durante el proceso. La importancia de la especificación de los requisitos software está demostrada por el hecho de que organizaciones profesionales como el IEEE y grandes clientes del sector del software, como el Departamento de Defensa de Estados Unidos han adoptado estándares para su redacción.

Desde la perspectiva del desarrollador del software, la especificación de requisitos software debería definir un objetivo bien establecido hacia el que el desarrollo software pueda dirigirse. Demasiado a menudo, sin embargo, el documento fracasa a la hora de proporcionar esta estabilidad. De hecho, la mayoría de los expertos en el campo de la ingeniería del software argumentan que una comunicación inadecuada y unos requisitos sometidos a variación son las principales causas de que el coste de los proyectos se dispare y de que los productos terminen entregándose tarde en el sector de la ingeniería del software. Pocos clientes insistirían en realizar cambios de importancia en el plano de un edificio una vez que se han construido los cimientos, pero son muy abundantes los casos de organizaciones que han ampliado o alterado de alguna manera las capacidades deseadas de un sistema software bastante después de que hubiera dado comienzo la construcción de ese software. Esto se puede deber a que la empresa ha decidido que el sistema que originalmente se estaba desarrollando tan solo para una filial debería en su lugar aplicarse a toda la empresa, o debido a que los avances en la tecnología han convertido en obsoletas las capacidades disponibles durante el análisis de requisitos inicial. En cualquier caso, lo que los ingenieros de software han comprobado es que es obligatoria una comunicación directa y frecuente con todas las partes interesadas en el proyecto.

IEEE

El Instituto de Ingenieros Eléctricos y Electrónicos (IEEE, *Institute of Electrical and Electronics Engineers*) es una organización de ingenieros eléctricos, electrónicos y de fabricación fundada en 1963 como resultado de la fusión del Instituto Americano de Ingenieros Eléctricos (fundado en 1884 por veinticinco ingenieros eléctricos, entre los que estaba Thomas Edison) y el Instituto de Ingenieros de Radio (fundado en 1912). Actualmente, el centro de operaciones del IEEE está en Piscataway, New Jersey. Este instituto incluye numerosas sociedades técnicas, como la Sociedad Aereoespacial y de Sistemas Electrónicos, la Sociedad de láseres y electroóptica, la Sociedad de Robótica y de Automatización, la Sociedad de Tecnología Vehicular y la Sociedad de Computadoras. Entre sus actividades, el IEEE está implicado en el desarrollo de estándares. Por ejemplo, los esfuerzos del IEEE condujeron a los estándares de punto flotante de precisión simple y de doble precisión (presentados en el Capítulo 1), que se emplean en la mayoría de las computadoras actuales.

Puede encontrar la página web del IEEE en la dirección `http://www.ieee.org`, la página web de la Sociedad de Computadoras del IEEE en `http://www.computer.org` y el Código de ética del IEEE en `http://www.ieee.org/about/whatis/code.html`.

Diseño Mientras que el análisis de requisitos proporciona una descripción del producto software propuesto, el diseño implica crear un plan para la construcción de ese sistema propuesto. En cierto sentido, el análisis de requisitos consiste en identificar el problema que hay que resolver, mientras que el diseño trata de desarrollar una solución para ese problema. Desde la perspectiva de un lego, el análisis de requisitos suele asociarse con decidir ***qué*** debe hacer un sistema software, mientras que el diseño se asocia con decidir ***cómo*** lo hará el sistema. Aunque esta descripción es esclarecedora, muchos ingenieros de software argumentan que es errónea, porque en realidad, durante el análisis de requisitos se toma también bastante en consideración el ***cómo***, mientras que durante el diseño se tiene también bastante en cuenta el ***qué***.

Es durante la etapa de diseño cuando se establece la estructura interna del sistema software. El resultado de la fase de diseño es una descripción detallada de la estructura del sistema software que puede convertirse en programas.

Si el proyecto consistiera en construir un edificio de oficinas en lugar de un sistema software, la etapa de diseño consistiría en desarrollar los planos estructurales detallados para un edificio que cumpla con los requisitos especificados. Por ejemplo, esos planos incluirían un conjunto de diagramas que describieran el edificio propuesto en diferentes niveles de detalle. Es a partir de esos documentos que se construiría el edificio real. Las técnicas para desarrollar esos planos han evolucionado a lo largo de los años e incluyen sistemas de notación estandarizados y numerosas metodologías de modelado y de generación de diagramas.

Del mismo modo, la generación de diagramas y el modelado desempeñan un papel importante en el diseño del software. Sin embargo, las metodologías y los sistemas de notación utilizados por los ingenieros de software no son tan estables como los utilizados en el campo de la arquitectura. Cuando se compara con la bien establecida disciplina de la arquitectura, la práctica de la ingeniería

del software parece enormemente dinámica, porque los investigadores tratan desesperadamente de encontrar mejores técnicas con las que acometer el desarrollo del software. Exploraremos estas arenas movedizas en la Sección 7.3 e investigaremos algunos de los sistemas de notación actuales y sus metodologías asociadas de modelado/generación de diagramas en la Sección 7.5.

Implementación La implementación implica la escritura de programas, la creación de archivos de datos y el desarrollo de bases de datos. Es durante la etapa de implementación cuando podemos ver la diferencia entre las tareas de un **analista de software** (que en ocasiones se denomina analista de sistemas) y un **programador**. El primero es una persona implicada en el proceso de desarrollo completo, quizá con un énfasis especial en las fases de análisis de requisitos y de diseño. El segundo es una persona implicada principalmente en el paso de implementación. En su interpretación más restringida, un programador está a cargo de la escritura de los programas que implementan el diseño creado por un analista de software. Habiendo hecho esta distinción, debemos recalcar de nuevo que no existe ninguna autoridad central que controle el uso de la terminología en todo el sector de la computación. Muchas personas que tienen el título de analista de software son esencialmente programadores y muchas que tienen el título de programador (o quizá de programador senior) son en realidad analistas de software en el pleno sentido del término. Esta terminología difusa se debe al hecho de que hoy día los pasos del proceso de desarrollo del software están a menudo entremezclados como pronto veremos.

Pruebas En la fase de desarrollo tradicional típica de hace años, las pruebas se equiparaban básicamente al proceso de depurar los programas y confirmar que el producto software final era compatible con la especificación de requisitos software. Sin embargo, actualmente esta visión de las pruebas se considera demasiado limitada. Los programas no son lo único que se prueba durante el proceso de desarrollo del software. De hecho, el resultado de cada paso intermedio del proceso de desarrollo completo debe ser "probado" para ver si es correcto. Además, como veremos en la Sección 7.6, las pruebas se consideran ahora solo como una de las tareas dentro del intento global del proceso del aseguramiento de la calidad del software, que es un objetivo que impregna todo el ciclo de vida del software. Por ello, muchos ingenieros de software argumentan que las pruebas no deberían ya contemplarse como un paso separado dentro del proceso de desarrollo, sino que en realidad deberían incorporarse, junto con sus múltiples manifestaciones, en los otros pasos, obteniéndose así un proceso de desarrollo en tres pasos cuyos componentes podrían denominarse algo así como "análisis de requisitos ***y confirmación***", "diseño ***y validación***" e "implementación ***y pruebas***".

Lamentablemente, incluso con las técnicas modernas de aseguramiento de la calidad, los sistemas software de gran envergadura continúan conteniendo errores, incluso después de realizar un considerable número de pruebas. Muchos de estos errores pueden no ser detectados durante toda la vida del sistema, pero otros pueden provocar graves fallos de funcionamiento. La eliminación de tales errores es uno de los objetivos de la ingeniería del software. El hecho de que sigan siendo prevalentes indica que todavía queda mucho por investigar.

Cuestiones y ejercicios

1. ¿Cómo afecta la etapa de desarrollo del ciclo de vida del software a la etapa de mantenimiento?
2. Resuma cada una de las cuatro etapas (análisis de requisitos, diseño, implementación y pruebas) de la fase de desarrollo del ciclo de vida del software.
3. ¿Qué papel desempeña una especificación de requisitos software?

7.3 Metodologías de ingeniería del software

Los primeros enfoques de la ingeniería del software insistían en realizar el análisis de requisitos, el diseño, la implementación y las pruebas de una manera estrictamente secuencial. La creencia era que existían demasiados riesgos durante el desarrollo de un sistema software como para permitir variaciones de esa pauta. Como resultado, los ingenieros de software insistían en que se completara toda la especificación de requisitos del sistema antes de comenzar con el diseño, y de la misma forma que se completara el diseño antes de iniciar la implementación. El resultado fue un proceso de desarrollo al que ahora denominamos **modelo en cascada**, por analogía con el hecho de que al proceso de desarrollo solo se le dejaba fluir en una dirección.

En años recientes, las técnicas de ingeniería del software han cambiado para reflejar la contradicción entre el entorno altamente estructurado impuesto por el modelo en cascada y el proceso mucho más libre de prueba y error, que tan vital resulta en ocasiones para la resolución creativa de problemas. Esto se refleja en la aparición del denominado **modelo incremental** de desarrollo software. De acuerdo con este modelo, el sistema software deseado se construye en incrementos, siendo el primer sistema una versión simplificada del producto final, con una funcionalidad limitada. Una vez probada esta versión (y una vez quizá evaluada por el futuro usuario) se añaden y prueban más características de una manera incremental hasta que el sistema está completo. Por ejemplo, si el sistema que se está desarrollando es un sistema de mantenimiento de registros de pacientes de un hospital, el primer incremento puede incorporar solo la capacidad de ver los registros de los pacientes, a partir de una pequeña muestra de todo el sistema de registros. Una vez que esta versión está operativa, se añaden paso a paso características adicionales, como la posibilidad de añadir nuevos registros y actualizar los existentes.

Otro modelo que representa una desviación con respecto al modelo en cascada es el **modelo iterativo**, que es similar al modelo incremental y en ocasiones se identifica con él, aunque se trata en realidad de dos modelos distintos. Mientras que el modelo incremental incorpora la noción de ***ampliar*** cada versión preliminar de un producto para conseguir una versión de mayor envergadura, el modelo iterativo se basa en el concepto de ***refinar*** cada versión. En realidad, el modelo incremental implica un proceso iterativo subyacente, mientras que el modelo iterativo por su parte puede añadir características de forma incremental.

Un ejemplo significativo de técnica iterativa es el denominado **proceso unificado racional** (RUP, *Rational Unified Process*) que fue creado por Rational Software Corporation, que ahora es una división de IBM. RUP es básicamente un paradigma de desarrollo software que redefine los pasos de la fase de desarrollo del ciclo de vida del software y proporciona directrices para llevar a cabo esos pasos. Esas directrices, junto con las herramientas CASE que les dan soporte, son comercializadas por IBM. Hoy día, RUP se aplica ampliamente en todo el sector del software. De hecho, su popularidad ha conducido al desarrollo de una versión no propietaria, denominada **proceso unificado**, que está disponible de forma no comercial.

Los modelos incremental e iterativo hacen uso en ocasiones de la tendencia que el campo del desarrollo software está experimentado hacia el **prototipado**, en el que se construyen y evalúan versiones incompletas del sistema propuesto denominadas prototipos. En el caso del modelo incremental, estos prototipos van evolucionando hacia el sistema final completo, un proceso conocido con el nombre de **prototipado evolutivo**. En una situación más iterativa, los prototipos pueden ser descartados en favor de una nueva implementación del diseño final. Esta técnica se conoce con el nombre de **prototipado descartable**. Un ejemplo que normalmente cae dentro de esta categoría del prototipado descartable es el **prototipado rápido**, en el que se construye rápidamente un ejemplo simple del sistema propuesto durante las etapas iniciales del desarrollo. Dicho prototipo puede consistir en solo unas pocas imágenes de pantalla que proporcionen una indicación de cómo interactuará el sistema con sus usuarios y de qué capacidades tendrá. El objetivo no es generar una versión funcional del producto, sino conseguir una herramienta de demostración que pueda utilizarse para clarificar la comunicación entre las partes implicadas en el proceso de desarrollo software. Por ejemplo, los prototipos rápidos de este estilo han demostrado ser ventajosos en la clarificación de los requisitos del sistema durante la fase de análisis de requisitos, o de ayuda durante las presentaciones de ventas a potenciales clientes.

Una encarnación menos formal de los conceptos incremental e iterativo que se ha estado utilizando durante años por parte de los entusiastas/aficionados de las computadoras es lo que se conoce con el nombre de **desarrollo de código fuente abierto**. Este es el medio por el que se produce buena parte del software libre que existe en la actualidad. Quizá el ejemplo más destacable sea el sistema operativo Linux, cuyo desarrollo de código fuente abierto fue dirigido originalmente por Linus Torvalds. El desarrollo de código fuente abierto de un paquete software tiene lugar de la forma siguiente: un único autor escribe una versión inicial del software (normalmente para satisfacer sus propias necesidades) y publica el código fuente y su documentación en Internet. Desde allí, puede ser descargado y utilizado por otros usuarios sin ningún coste. Puesto que estos otros usuarios disponen del código fuente y de la documentación, pueden modificar o mejorar el software para que se ajuste a sus propias necesidades o para corregir los errores que encuentren. Esos usuarios informan de las modificaciones realizadas al autor original, quien las incorpora a la versión publicada del software, haciendo que esta versión ampliada esté disponible para posteriores modificaciones. En la práctica, es posible de esta manera que un paquete software evolucione a través de varias ampliaciones sucesivas en una única semana.

Quizá la variación más pronunciada respecto al modelo en cascada está representada por el conjunto de metodologías conocido con el nombre de **métodos ágiles**, cada uno de los cuales propone: una implementación rápida y temprana basada en el concepto incremental, una adecuada capacidad de respuesta a las variaciones en los requisitos y un menor énfasis en la rigurosidad del análisis de requisitos y en el diseño. Un ejemplo de método ágil es la denominada **programación extrema** (XP, *eXtreme Programming*). De acuerdo con el modelo XP, un equipo formado por menos de una docena de personas desarrolla el software trabajando en un espacio común donde comparten libremente sus ideas y se ayudan entre sí en el proyecto de desarrollo. El software se desarrolla incrementalmente por medio de ciclos diarios repetidos de análisis de requisitos, diseño, implementación y pruebas, todos ellos de carácter informal. Así, pueden ir apareciendo de manera periódica nuevas versiones ampliadas del paquete software, cada una de las cuales puede ser evaluada por cada una de las partes interesadas en el proyecto y cada una de las cuales puede ser usada también para tratar de identificar futuras mejoras incrementales. En resumen, los métodos ágiles se caracterizan por la flexibilidad, que contrasta enormemente con ese modelo en cascada que evoca la imagen de gerentes y programadores trabajando cada uno en su propio despacho, mientras llevan a cabo de una manera rígida determinadas partes bien definidas de la tarea global de desarrollo del software.

Las diferencias que pueden percibirse al comparar el modelo en cascada y el modelo XP revelan la gran variedad de metodologías que se aplican actualmente al proceso de desarrollo software con la esperanza de encontrar mejores formas de construir software fiable de una manera eficiente. Las investigaciones en este campo siguen siendo muy intensas, y aunque se están haciendo progresos, todavía queda mucho trabajo por hacer.

Cuestiones y ejercicios

1. Resuma la diferencia entre el modelo en cascada tradicional para el modelo de desarrollo de software y los más recientes paradigmas incremental e iterativo.
2. Identifique tres paradigmas de desarrollo que representen una variación con respecto al modelo en cascada tradicional.
3. ¿Cuál es la diferencia entre el prototipado evolutivo tradicional y el desarrollo de código fuente abierto?
4. ¿Qué problemas potenciales podrían surgir en lo que respecta a los derechos de propiedad del software desarrollado mediante la metodología de código fuente abierto?

7.4 Modularidad

Un punto clave que hemos señalado en la Sección 7.2 es que para modificar software es preciso entender el programa o al menos las partes relevantes del mismo. Conseguir esa comprensión suele ser ya bastante difícil en el caso de

pequeños programas y sería prácticamente imposible si tratamos con sistemas software grandes, salvo que recurramos a la **modularidad**; es decir, a la división del software en unidades más manejables, que se denominan de forma genérica **módulos**, cada una de las cuales trata únicamente con una parte de las tareas globales asignadas al software.

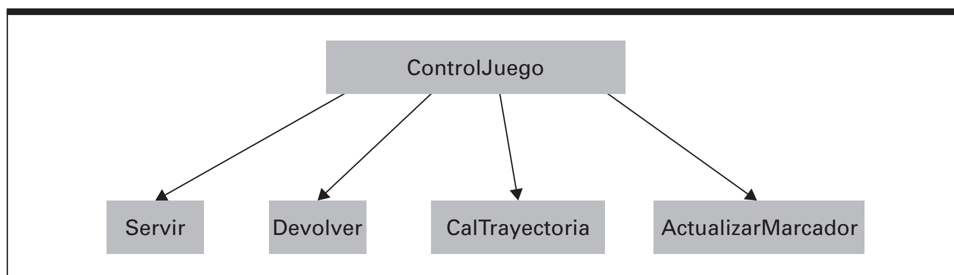
Implementación modular

Los módulos se presentan de diversas maneras. Ya hemos visto (Capítulos 5 y 6) que en el contexto del paradigma imperativo, los módulos aparecen como procedimientos. Por el contrario, el paradigma orientado a objetos emplea los objetos como módulos constituyentes básicos. Estas distinciones son importantes porque determinan el objetivo subyacente en el inicio del proceso de diseño del software. ¿Es el objetivo representar la tarea global en forma de procesos individuales manejables o identificar los objetos del sistema y comprender cómo interactúan?

Para ilustrar esto, consideremos cómo podría progresar el proceso de desarrollo de un programa modular simple para simular un juego de tenis en los paradigmas imperativo y de orientación de objetos. En el paradigma imperativo, comenzaríamos considerando las acciones que deben tener lugar. Puesto que cada punto comienza con uno de los jugadores teniendo el servicio, podemos empezar considerando un procedimiento denominado `Servir` que (basándose en las características del jugador y quizá un poco de aletoriedad) calcularía la velocidad y dirección iniciales de la bola. Después, necesitaríamos determinar la trayectoria de la bola (¿Chocará con la red? ¿Hacia dónde rebotará?). Podemos decidir incluir estos cálculos en otro procedimiento denominado `CalTrayectoria`. El siguiente paso podría consistir en determinar si el otro jugador podrá devolver la bola y, en caso afirmativo, tendremos que calcular la nueva dirección y velocidad de la bola. Podemos incluir estos cálculos en otro procedimiento denominado `Devolver`.

Continuando de esta manera, podríamos llegar a la estructura modular ilustrada por el **diagrama de estructura** mostrado en la Figura 7.3, en el que los procedimientos se representan mediante rectángulos y las dependencias entre procedimientos (implementadas por llamadas a procedimientos) se representan mediante flechas. En particular, el diagrama indica que todo el juego está controlado por un procedimiento denominado `ControlJuego` y que para realizar su tarea, `ControlJuego` invoca los servicios de los procedimientos `Servir`, `Devolver`, `CalTrayectoria` y `ActualizarMarcador`.

Figura 7.3 Un diagrama de estructura simple.



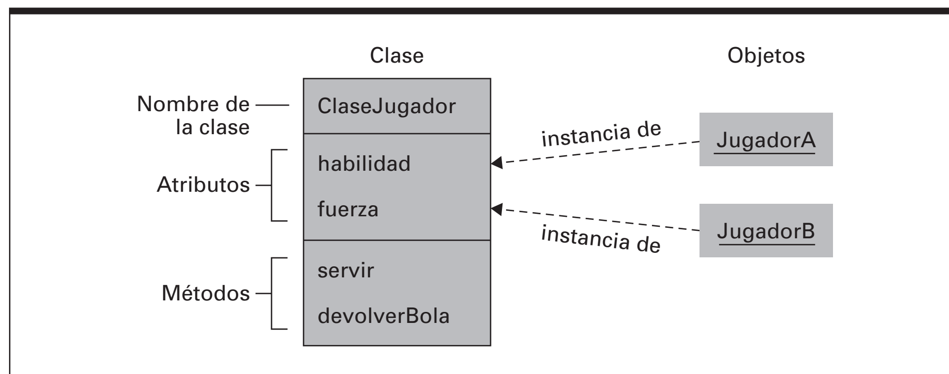
Observe que el diagrama de estructura no indica cómo debe realizar su tarea cada procedimiento. En lugar de ello, simplemente identifica los procedimientos e indica las dependencias entre ellos. En realidad, el procedimiento `ControlJuego` podría realizar su tarea llamando primero al procedimiento `Servir`, luego llamando repetidamente a los procedimientos `CalTrayectoria` y `Devolver` hasta que uno de ellos informe de que se ha producido un fallo y finalmente invocando los servicios de `ActualizarMarcador`, antes de repetir todo el proceso invocando de nuevo a `Servir`.

En esta etapa, hemos obtenido únicamente un esbozo muy simple del programa deseado, pero nos sirve para dejar claro lo que queremos decir. De acuerdo con el paradigma imperativo, hemos estado diseñando el programa considerando las actividades que hay que realizar, debido a lo cual estamos obteniendo un diseño en el que los módulos son los procedimientos.

Consideremos ahora el diseño del programa, pero esta vez en el contexto del paradigma de orientación a objetos. Nuestra primera idea sería que hay dos jugadores que debemos representar mediante dos objetos: `JugadorA` y `JugadorB`. Estos objetos tendrán la misma funcionalidad, pero diferentes características (ambos deben ser capaces de servir y de devolver las bolas, pero pueden hacerlo con diferente fuerza y diferentes habilidades). Por tanto, estos objetos serán instancias distintas de una misma clase (recuerde que en el Capítulo 6 hemos presentado el concepto de clase: una plantilla que define los procedimientos, denominados métodos, y atributos, denominados variables de instancia, que hay que asociar con cada objeto). Esta clase, que denominaremos `ClaseJugador`, contendrá los métodos `servir` y `devolver` que simularán las acciones correspondientes del jugador. También contendrá atributos (como `habilidad` y `fuerza`) cuyos valores reflejarán las características del jugador. Nuestro diseño hasta el momento está representado por el diagrama mostrado en la Figura 7.4. En él podemos ver que `JugadorA` y `JugadorB` son instancias de la clase `ClaseJugador` y que esta clase contiene los atributos `habilidad` y `fuerza`, así como los métodos `servir` y `devolverBola`.

A continuación necesitamos un objeto que haga el papel del árbitro que determina si las acciones realizadas por los jugadores son legales. Por ejemplo, ¿ha podido el servicio pasar por encima de la red y tocar tierra en el área apropiada de la pista? Con este propósito, podemos definir un objeto denominado `Juez` que contenga los métodos `evaluarServicio` y `evaluarDevolucion`. Si el

Figura 7.4 La estructura de ClaseJugador y sus instancias.



objeto Juez determina que un servicio o una devolución son aceptables, el juego continúa. En caso contrario, el Juez envía un mensaje a otro objeto denominado Marcador para registrar el resultado de la forma pertinente.

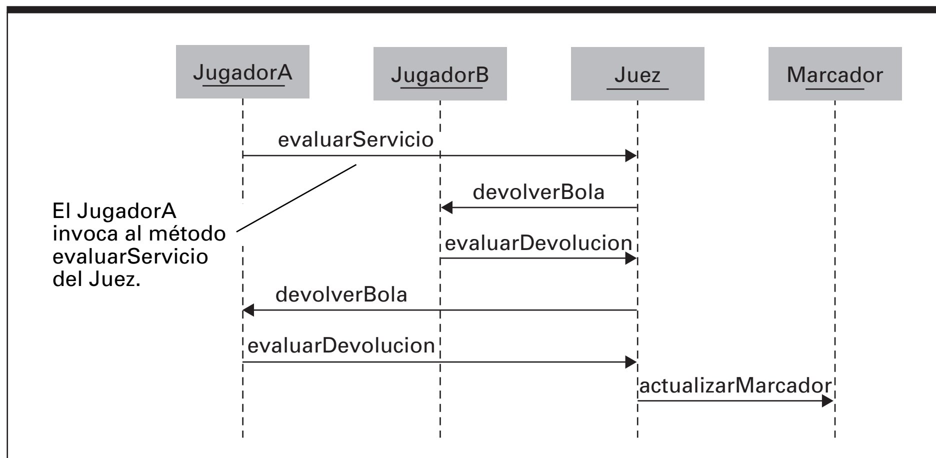
En este punto, el diseño de nuestro programa de tenis constará de cuatro objetos: JugadorA, JugadorB, Juez y Marcador. Para clarificar nuestro diseño, consideremos la secuencia de sucesos que pueden tener lugar mientras se juega una bola, como se ilustra en la Figura 7.5, en la que hemos representado los objetos implicados como rectángulos. La figura trata de ilustrar la comunicación entre estos objetos como resultado de invocar el método servir dentro del objeto JugadorA. Los sucesos aparecen cronológicamente de arriba hacia abajo. Tal como muestra la primera flecha horizontal, el JugadorA informa de su servicio al objeto Juez llamando al método evaluarServicio. El Juez determina entonces que el servicio es bueno y pide al JugadorB que devuelva la bola invocando al método devolverBola del JugadorB. El punto termina cuando el Juez determina que el JugadorA ha fallado y pide al objeto Marcador que anote el resultado.

Como en el caso de nuestro ejemplo imperativo, por el momento, nuestro programa orientado a objetos es todavía muy simple. Sin embargo, hemos progresado lo suficiente como para ver que el paradigma de orientación a objetos conduce a un diseño modular en el que los componentes fundamentales son objetos.

Acoplamiento

Hemos presentado la modularidad como una forma de obtener un software manejable. La idea es que cualquier modificación futura solo tendrá que aplicarse a unos pocos módulos, permitiendo que la persona que efectúe la modificación se concentre en esa parte del sistema, en lugar de tener que pelearse con el paquete completo. Esto depende, por supuesto, de la suposición de que los cambios realizados en un módulo no afectarán de manera imprevista a otros módulos del sistema. En consecuencia, un objetivo a la hora de diseñar un sistema modular debería ser maximizar la independencia entre módulos o, en

Figura 7.5 La interacción entre objetos resultante de un servicio del JugadorA.



otras palabras, minimizar el vínculo entre módulos (lo que se conoce con el nombre de **acoplamiento** intermodular). De hecho, una métrica que se emplea en ocasiones para medir la complejidad de un sistema software (y obtener así un método de estimar los gastos de mantenimiento del software) consiste en medir su acoplamiento intermodular.

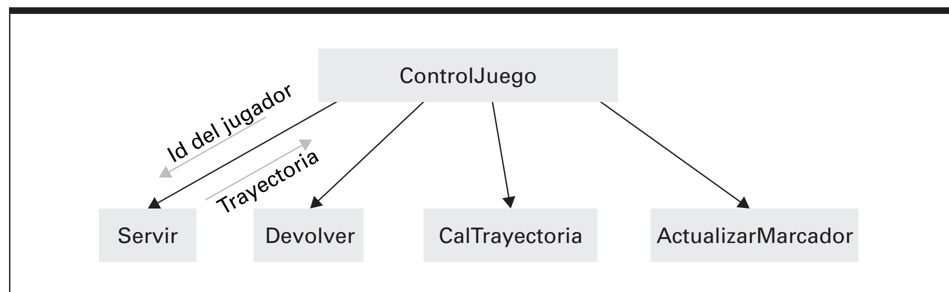
El acoplamiento intermodular se presenta de diversas formas. Una de ellas es el **acoplamiento de control**, que tiene lugar cuando un módulo pasa el control de la ejecución a otro, como sucede por ejemplo en una llamada a procedimiento. El diagrama de estructura de la Figura 7.3 representa el acoplamiento de control que existe entre los distintos procedimientos de ese programa. En particular, la flecha que va del módulo `ControlJuego` a `Servir` indica que el primero de esos módulos pasa el control al segundo. También es acoplamiento de control lo que se representa en la Figura 7.5, en la que las flechas trazan la ruta que sigue el control al pasar de un objeto a otro.

Otra forma de acoplamiento intermodular es el **acoplamiento de datos**, que hace referencia a la compartición de datos entre módulos. Si dos módulos interactúan con el mismo elemento de datos, entonces las modificaciones realizadas por un módulo pueden afectar al otro, y las modificaciones en el formato de los propios datos podría tener repercusiones en ambos módulos.

El acoplamiento de datos entre procedimientos puede producirse de dos formas distintas. Una de ellas es pasando datos explícitamente de un procedimiento a otro en forma de parámetros. Este tipo de acoplamiento se representa en un diagrama de estructura mediante una flecha entre los procedimientos, que se etiqueta para indicar los parámetros que se están pasando. La dirección de la flecha indica la dirección en la que se transfiere el elemento. Por ejemplo, la Figura 7.6 es una versión ampliada de la Figura 7.3 en la que hemos indicado que el procedimiento `ControlJuego` le indicará al procedimiento `Servir` qué características del jugador hay que simular cuando invoque `Servir` y que el procedimiento `Servir` informará de la trayectoria de la bola a `ControlJuego` cuando `Servir` haya completado su tarea.

Un acoplamiento de datos similar es el que se produce entre los objetos en un diseño orientado a objetos. Por ejemplo, cuando el `JugadorA` pide al objeto `Juez` que evalúe su servicio (véase la Figura 7.5), le tiene que pasar al `Juez` la información de la trayectoria. Por otro lado, una de las ventajas del paradigma de orientación a objetos es que tiende inherentemente a reducir a un mínimo el acoplamiento de datos entre objetos. Esto se debe a que los métodos dentro de un objeto tienden a incluir todos aquellos procedimientos que manipulen

Figura 7.6 Un diagrama de estructura que incluye acoplamiento de datos.



los datos internos del objeto. Por ejemplo, el objeto `JugadorA` contendrá información relativa a las características de ese jugador, así como todos los métodos que requieran dicha información. A su vez, no hay ninguna necesidad de pasar dicha información a otros objetos, con lo que el acoplamiento de datos entre objetos se minimiza.

A diferencia del paso de datos de manera explícita en forma de parámetros, los datos también pueden compartirse entre módulos de manera implícita en la forma de **datos globales**, que son elementos de datos que están disponibles automáticamente para todos los módulos de un sistema, a diferencia de los elementos de datos locales que solo están accesibles dentro de un módulo concreto, a menos que se pasen explícitamente a otro. La mayoría de los lenguajes de alto nivel proporcionan formas de implementar tanto datos locales como globales, pero los datos globales deben utilizarse con precaución. El problema es que una persona que intente modificar un módulo que depende de datos globales puede encontrar muy difícil identificar cómo interactúa el módulo en cuestión con los demás módulos. En pocas palabras, el uso de datos globales puede reducir la utilidad del módulo como herramienta abstracta.

Cohesión

Tan importante como minimizar el acoplamiento entre módulos es maximizar el acoplamiento interno dentro de cada módulo. Se utiliza el término **cohesión** para hacer referencia a este acoplamiento interno o, en otras palabras, al grado de relación entre las distintas partes internas de un módulo. Para apreciar la importancia de la cohesión, debemos ir más allá del desarrollo inicial de un sistema y tener en cuenta todo el ciclo de vida del software. Si llega a ser necesario efectuar cambios en un módulo, la existencia dentro de él de diversas actividades relacionadas entre sí puede terminar haciendo confuso lo que de otro modo sería un proceso simple. Por ello, además de tratar de conseguir un acoplamiento débil entre módulos, los diseñadores software tratan de lograr una alta cohesión intramodular.

Una forma débil de cohesión es la que se conoce como **cohesión lógica**. Es la cohesión dentro de un módulo inducida por el hecho de que sus elementos internos realizan actividades que tienen una naturaleza lógica similar. Por ejemplo, considere un módulo que se encarga de realizar todas las comunicaciones de un sistema con el mundo exterior. El "adhesivo" que mantiene la cohesión de ese módulo es que todas las actividades del mismo están relacionadas con las comunicaciones. Sin embargo, los temas de esas comunicaciones pueden variar enormemente. Algunas de esas comunicaciones pueden estar dirigidas a obtener datos, mientras que otras pueden dedicarse a informar de los resultados.

Otro tipo más fuerte de cohesión es la que se conoce con el nombre de **cohesión funcional**, que quiere decir que todas las partes del módulo están centradas en realizar una única actividad. En un diseño imperativo, la cohesión funcional puede a menudo incrementarse aislando las subtareas en otros módulos y luego utilizando esos módulos como herramientas abstractas. Esto se ilustra en nuestra simulación del ejemplo del tenis (véase de nuevo la Figura 7.3), en el que el módulo `ControlJuego` utiliza a los demás módulos como herramientas abstractas, para poder concentrarse en controlar el juego, en lugar de

distraerse con los detalles del servicio, de la devolución de las bolas y del mantenimiento del marcador.

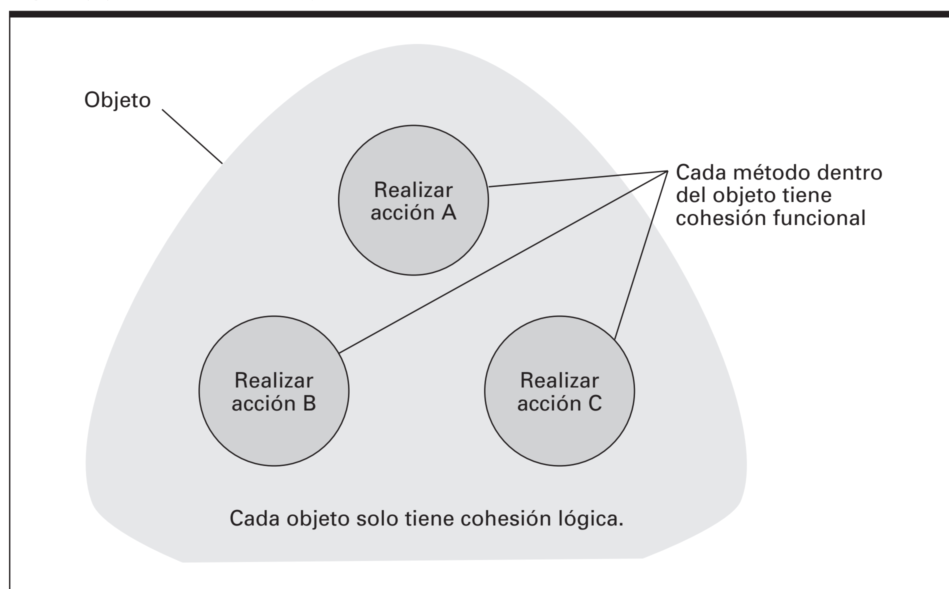
En los diseños orientados a objetos, cada objeto completo suele tener únicamente una cohesión lógica, porque los métodos incluidos dentro del objeto realizan a menudo actividades que solo están débilmente relacionadas, siendo el único nexo de unión el hecho de que se trata de actividades realizadas por un mismo objeto. Así, en nuestro ejemplo de simulación de tenis, cada objeto jugador contenía métodos tanto para servir como para devolver la bola, que son actividades significativamente distintas. Dicho tipo de objeto solo tendría, por tanto, una cohesión lógica. Sin embargo, los diseñadores de software deben tratar de hacer que cada método individual dentro de un objeto presente una cohesión funcional. Es decir, aunque el objeto solo tenga cohesión lógica, cada método dentro de él debe llevar a cabo una única tarea que tenga una cohesión funcional (Figura 7.7).

Ocultamiento de la información

Una de las piedras angulares de un buen diseño modular es la que está plasmada en el concepto de **ocultación de la información**, que hace referencia a la acción de restringir la información a una parte específica de un sistema software. Aquí el término ***información*** debe interpretarse en un sentido amplio, incluyendo cualquier conocimiento acerca de la estructura y del contenido de una unidad de programa. Desde ese punto de vista, incluye los datos, el tipo de estructuras de datos utilizadas, los sistemas de codificación, la estructura interna de un módulo, la estructura lógica de una unidad procedimental y cualesquiera otros factores relativos a las propiedades internas de un módulo.

El objetivo del ocultamiento de la información es evitar que las acciones de los módulos tengan dependencias o efectos innecesarios en otros módulos. En

Figura 7.7 Cohesión lógica y funcional dentro de un objeto.



caso contrario, la validez de un módulo puede verse comprometida, quizá por errores en el desarrollo de otros módulos o por modificaciones desacertadas durante la etapa de mantenimiento del software. Por ejemplo, si un módulo no restringe el uso de sus datos internos por parte de otros módulos, entonces dichos datos podrían verse corrompidos por esos otros módulos. O bien, si un módulo está diseñado para aprovecharse de la estructura interna de otro, podría llegar a funcionar incorrectamente más adelante en caso de que esa estructura interna de la que depende sea modificada.

Es importante dejar claro que el ocultamiento de información tiene dos caras complementarias: una como objetivo de diseño y otra como objetivo de implementación. Un módulo debe diseñarse de manera que otros módulos no necesiten acceder a su información interna y cada módulo debe implementarse de forma que se refuercen las fronteras del módulo. Como ejemplo de la primera de estas dos caras tendríamos los esfuerzos para maximizar la cohesión y minimizar el acoplamiento. Como ejemplos de la segunda tendríamos el uso de variables locales, la técnica de encapsulación y el uso de estructuras de control bien definidas.

Finalmente, hay que recalcar que el ocultamiento de información es crucial de cara a la abstracción y al uso de herramientas abstractas. De hecho, el concepto de herramienta abstracta es precisamente el de una "caja negra" cuya estructura interior puede ser ignorada por su usuario, permitiéndole asi concentrarse en la aplicación de mayor envergadura que tenga entre manos. Por tanto, en este sentido, el ocultamiento de información se corresponde con el concepto de "sellar" la herramienta abstracta de forma bastante similar a como podría utilizarse una carcasa hermética para proteger equipos electrónicos complejos y potencialmente peligrosos. Tanto el ocultamiento de información como esa carcasa hermética protegen a sus usuarios de los peligros que hay en su interior, además de proteger su interior frente a la manipulación por parte de los usuarios.

Componentes

Ya hemos mencionado que uno de los obstáculos en el campo de la ingeniería del software es la falta de bloques componentes prefabricados a partir de los cuales construir sistemas software de mayor tamaño. El enfoque modular de desarrollo software proporciona algunas esperanzas en lo que a esto respecta. En particular, el paradigma de programación orientada a objetos está demostrando ser especialmente útil, porque los objetos forman unidades completas y autocontenidas, que tienen interfaces claramente definidas con su entorno. Una vez que se ha diseñado un objeto (o, para ser más precisos, una clase) para jugar un cierto papel, se le puede utilizar para desempeñar ese papel en cualquier programa que requiera ese tipo de servicio. Además, la herencia proporciona un medio de refinar las definiciones prefabricadas de objetos en aquellos casos en los que las definiciones tengan que ser personalizadas, con el fin de adaptarse a las necesidades de una aplicación específica.

Por eso, no es sorprendente que los lenguajes de programación orientados a objetos C++, Java y C# vayan acompañados por conjuntos de "plantillas" prefabricadas, a partir de las cuales los programadores pueden implementar objetos fácilmente para que desempeñen determinados papeles. En particular,

C++ está asociado con la librería de plantillas estándar STL (*Standard Template Library*) de C++, el entorno de programación Java está acompañado por la interfaz de programación de aplicaciones (API, *Application Programming Interface*) de Java, y los programadores de C# tienen acceso a la librería de clases del entorno .NET.

El hecho de que los objetos y las clases tengan el potencial de proporcionar componentes prefabricados para el diseño software no quiere decir que sean ideales. Un problema es que proporcionan bloques relativamente pequeños entre los que elegir. Así, un objeto es en realidad un caso especial del concepto más general de **componente**, que es por definición una unidad reutilizable de software. En la práctica, la mayoría de los componentes están basados en el paradigma orientado a objetos y se presentan en la forma de un conjunto de uno o más objetos que funcionan como unidad autocontenida.

La investigación en el desarrollo y la utilización de componentes ha conducido a la aparición de un campo emergente que se conoce con el nombre de **arquitectura de componentes** (también denominado ingeniería software basada en componentes) en la que el papel tradicional de un programador se sustituye por un **ensamblador de componentes**, que construye sistemas software a partir de componentes prefabricados que, en muchos entornos de desarrollo, se muestran como iconos en una interfaz gráfica. En lugar de dedicarse a la programación interna de los componentes, la metodología de un ensamblador de componentes consiste en seleccionar los componentes adecuados a partir de conjuntos de componentes predefinidos y luego interconectarlos, con una mínima personalización para obtener la funcionalidad deseada. De hecho, una propiedad de un componente bien diseñado es que pueda ampliarse para

Ingeniería del software en el mundo real

El siguiente escenario es típico de los problemas con los que los ingenieros de software se encuentran en el mundo real. Imagine que la empresa XYZ contrata a una consultoría de ingeniería software para desarrollar e instalar un sistema software integrado de ámbito corporativo, con el fin de manejar las necesidades de procesamiento de datos de la empresa. Como parte del sistema de la empresa XYZ, se utiliza una red de equipos PC para proporcionar a los empleados acceso al sistema corporativo. Así, cada empleado dispone de un PC en su puesto de trabajo. Pronto estos PC no solo se emplean para acceder al nuevo sistema de gestión de datos, sino como herramienta personalizable con la que cada empleado incrementa su productividad. Por ejemplo, un empleado puede desarrollar un programa de hoja de cálculo que le ayude a llevar a cabo sus tareas de manera más eficiente. Lamentablemente, esas aplicaciones personalizadas pueden no estar bien diseñadas o no haber sido suficientemente probadas, y pueden integrar características que el empleado no termine de comprender. A medida que van pasando los años, el uso de esas aplicaciones ad hoc pasa a formar parte integrante de los procedimientos operativos internos de la empresa. Además, los empleados que desarrollaron esas aplicaciones pueden haber sido ascendidos, transferidos o pueden haber abandonado la empresa, dejando atrás a otros empleados que usan un programa que no comprenden. El resultado es que lo que comenzó como un sistema coherente y bien diseñado puede llegar a depender de un auténtico puzzle de aplicaciones mal diseñadas, no documentadas y proclives a errores.

integrar nuevas características para una aplicación concreta sin necesidad de realizar modificaciones internas.

Un área en la que las arquitecturas de componentes han encontrado terreno fértil es en los sistemas para teléfonos inteligentes. Debido a las restricciones de recursos en estos dispositivos, las aplicaciones son en la práctica un conjunto de componentes que colaboran entre sí, proporcionando cada uno de ellos una determinada función para la aplicación. Por ejemplo, cada pantalla de visualización de una aplicación suele ser un componente separado. Entre bastidores, pueden existir otros componentes de servicio para almacenar y leer información en una tarjeta de memoria, realizar algún tipo de función continua, como por ejemplo reproducir música, o acceder a información a través de Internet. Cada uno de estos componentes se inicializa y se detiene individualmente, según se vaya requiriendo para dar servicio al usuario de forma eficiente; sin embargo, la aplicación se presenta como una serie integrada de pantallas y acciones.

Además de que exista la motivación de limitar el uso de los recursos del sistema, la arquitectura de componentes de los teléfonos inteligentes es ventajosa a la hora de integrar aplicaciones. Por ejemplo, Facebook (una red social muy conocida) puede, al ejecutarse en un teléfono inteligente, usar los componentes de la aplicación de contactos para agregar como contactos todos los amigos de Facebook. Además, la aplicación de telefonía (la que se encarga de gestionar las funciones del teléfono) puede también acceder a los componentes de contactos para detectar quién es el llamante cuando entra una llamada. Así, al recibir la llamada de un amigo de Facebook, la imagen de ese amigo puede visualizarse en la pantalla del teléfono (junto con su último mensaje en Facebook).

Cuestiones y ejercicios

1. ¿En qué se diferencia una novela de una enciclopedia en términos del grado de acoplamiento entre sus distintas unidades, como por ejemplo capítulos, secciones o entradas? ¿Y qué sucede con la cohesión?
2. Un evento deportivo suele estar dividido en una serie de unidades. Por ejemplo, un partido de baloncesto está dividido en cuartos y uno de tenis está dividido en sets. Analice el acoplamiento entre esos “módulos”. ¿En qué sentido tienen cohesión esas unidades?
3. ¿Cree que el objetivo de maximizar la cohesión es compatible con minimizar el acoplamiento? En otras palabras, si la cohesión se incrementa, ¿el acoplamiento tiende de forma natural a reducirse?
4. Defina los conceptos de acoplamiento, cohesión y ocultamiento de información.
5. Amplíe el diagrama de estructura de la Figura 7.3 para incluir el acoplamiento de datos entre los módulos `Controljuego` y `ActualizarMarcador`.
6. Dibuje un diagrama similar al de la Figura 7.5 para representar la secuencia que se produciría si se declarara inválido el servicio del `JugadorA`.

7. ¿Cuál es la diferencia entre un programador tradicional y un ensamblador de componentes?
8. Suponiendo que la mayoría de los teléfonos inteligentes disponen de una serie de aplicaciones de organización personal (calendario, contactos, relojes, redes sociales, sistemas de correo electrónico, mapas, etc.), ¿que combinaciones de funciones componentes le parece que serían útiles e interesantes?

7.5 Herramientas existentes

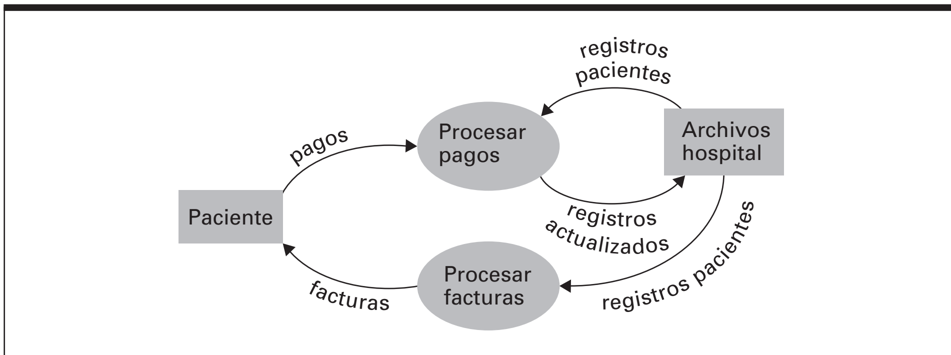
En esta sección investigaremos algunas de las técnicas de modelado y de los sistemas de notación utilizados durante las etapas de análisis y diseño del desarrollo de software. Varias de estas técnicas y sistemas fueron desarrollados durante los años en los que el paradigma imperativo dominaba el campo de la ingeniería del software. De ellos, algunos han encontrado un papel útil en el contexto del paradigma de la orientación a objetos, mientras que otros, como el diagrama de estructura (véase de nuevo la Figura 7.3), son específicos del paradigma imperativo. Vamos a comenzar considerando algunas de las técnicas que han sobrevivido a partir de sus raíces imperativas y luego continuaremos explorando otras herramientas orientadas a objetos más recientes. También abordaremos el papel creciente de los patrones de diseño.

Algunos viejos conocidos

Aunque el paradigma imperativo trata de construir el software en términos de procedimientos, una forma de identificar dichos procedimientos es considerando los datos que hay que manipular en lugar de los propios procedimientos. La teoría es que estudiando cómo se mueven los datos a través de un sistema, se pueden identificar aquellos puntos en los que se modifica el formato de los datos o en los que las rutas de datos convergen o se subdividen. A su vez, esas serán las ubicaciones en las que tendrá lugar el procesamiento, por lo que el análisis del flujo de datos conduce a la identificación de los procedimientos. Un **diagrama de flujo** es un medio de representar la información obtenida a partir de dichos análisis del flujo de datos. En un diagrama de este tipo, las flechas representan flujos de datos, los óvalos representan puntos en los que tiene lugar la manipulación de esos datos y los rectángulos representan orígenes y almacenes de datos. Por ejemplo, la Figura 7.8 muestra un diagrama de flujo de datos elemental, que ilustra el sistema de facturación a pacientes de un hospital. Observe que el diagrama muestra que los `Pagos` (que fluyen desde los pacientes) y los `RegistrosPaciente` (que fluyen desde los archivos del hospital) se combinan en el óvalo `ProcesarPagos` desde el que los `RegistrosActualizados` fluyen de vuelta a los archivos del hospital.

Los diagramas de flujo de datos no solo ayudan a identificar los procedimientos durante la etapa de diseño del desarrollo software, sino que también resultan útiles a la hora de comprender el sistema propuesto durante la etapa de análisis. De hecho, la construcción de diagramas de flujos de datos puede servir como medio para mejorar la comunicación entre los clientes y los ingenieros de software (cuando el ingeniero de software trata de comprender qué

Figura 7.8 Un diagrama de flujo de datos simple.



es lo que quiere el cliente y el cliente trata de describir sus expectativas), por lo que estos diagramas continúan encontrando aplicaciones aún cuando el paradigma imperativo haya perdido popularidad.

Otra herramienta que los ingenieros de software han estado utilizando durante años es el **diccionario de datos**, que es un repositorio central de información acerca de los elementos de datos que aparecen a través de un sistema software. Esta información incluye el identificador utilizado para hacer referencia a cada elemento, las indicaciones acerca de qué es lo que constituye una entrada válida en cada elemento (¿el elemento será siempre numérico o quizá será siempre alfabético?, ¿ cuál será el rango de valores que puede asignarse a ese elemento?), el lugar en el que se almacena el elemento (¿se almacenará ese elemento en un archivo o en una base de datos? ¿En cuál?) y dónde se hace referencia al elemento dentro del software (¿qué módulos requerirán la información representada por este elemento?).

Uno de los objetivos a la hora de construir un diccionario de datos consiste en mejorar la comunicación entre todas las partes interesadas en un sistema software y el ingeniero de software encargado de la tarea de convertir las necesidades de esas partes interesadas en una especificación de requisitos. En este contexto, la construcción de un diccionario de datos ayuda a garantizar, por ejemplo, que se detecte durante la etapa de análisis el hecho de que los códigos de serie de las piezas fabricadas no son realmente numéricos, en lugar de descubrirlo posteriormente durante las etapas de diseño o de implementación. Otro objetivo asociado con el diccionario de datos es garantizar la uniformidad en todo el sistema. Normalmente, es gracias a la construcción del diccionario que pueden detectarse las redundancias y las contradicciones. Por ejemplo, el elemento denominado `NumeroComponente` en los registros de inventario puede ser el mismo que el elemento `IdComponente` en los registros de ventas. O bien, el departamento de personal puede utilizar el elemento `Nombre` para hacer referencia a un empleado, mientras que los registros de inventario pueden contener el término `Nombre` para referirse a un componente.

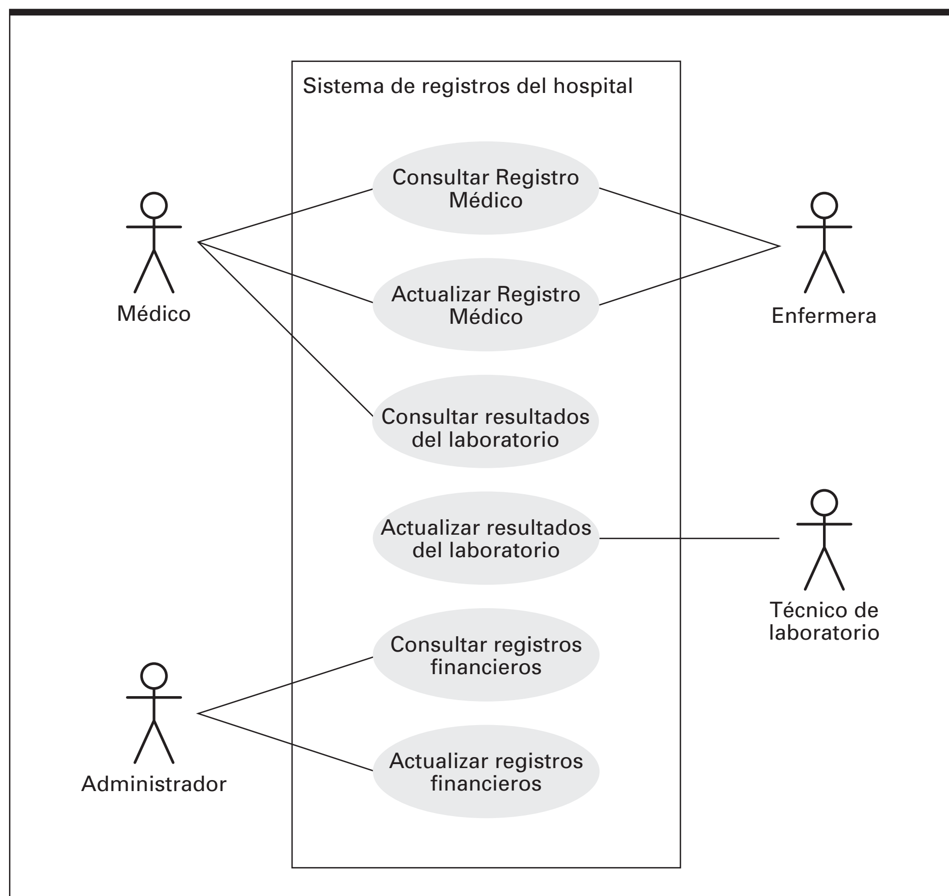
UML

Los diagramas de flujo de datos y los diccionarios de datos eran herramientas del arsenal de la ingeniería del software mucho antes de la aparición del paradigma de orientación a objetos y han continuado encontrando un papel útil que

desempeñar, aún cuando el paradigma imperativo para el que fueron originalmente desarrollados sea ahora menos popular. Volvamos nuestra atención sobre un conjunto de herramientas más moderno, conocido con el nombre de **UML** (*Unified Modeling Language*, Lenguaje unificado de modelado) que fue desarrollado teniendo en mente el paradigma de orientación a objetos. Sin embargo, la primera herramienta de este conjunto que vamos a considerar es útil independientemente de cuál sea el paradigma subyacente que se utilice, ya que simplemente trata de capturar la imagen del sistema propuesto desde el punto de vista del usuario. Esta herramienta es el **diagrama de casos de uso**. En la Figura 7.9 se muestra un ejemplo de este tipo de diagrama.

Un diagrama de casos de uso muestra el sistema propuesto como un rectángulo de gran tamaño, en el que las interacciones (denominadas **casos de uso**) entre el sistema y sus usuarios se representan como óvalos y los usuarios del sistema (llamados **actores**) se representan mediante figuras de personas (aún cuando un actor puede no ser una persona). Así, el diagrama de la Figura 7.9 indica que el `Sistema de registros del hospital` propuesto será utilizado tanto por los `Médicos` como por las `Enfermeras` para `Consultar Registros Médicos`.

Figura 7.9 Un diagrama de casos de uso simple.



Mientras que los diagramas de casos de uso contemplan desde el exterior el sistema software propuesto, UML ofrece diversas herramientas para representar el diseño interno orientado a objetos del sistema. Una de esas herramientas es el **diagrama de clases**, que es un sistema de notación que permite representar la estructura de las clases y las relaciones existentes entre ellas (denominadas **asociaciones** en la jerga de UML). Por ejemplo, considere la relación entre médicos, pacientes y habitaciones del hospital. Vamos a suponer que los objetos que representan dichas entidades se construyen a partir de las clases `Medico`, `Paciente` y `Habitacion`, respectivamente.

La Figura 7.10 muestra cómo las relaciones entre estas clases podrían representarse mediante un diagrama de clases UML. Las clases están representadas por rectángulos y las relaciones mediante líneas. Las líneas de asociación puede o no estar etiquetadas. Si lo están, puede utilizarse una punta de flecha rellena para indicar la dirección en la que hay que leer la etiqueta. Por ejemplo, en la Figura 7.10, la flecha situada a continuación de la etiqueta `cuida a` indica que un médico `cuida a` un paciente en lugar de ser el paciente el que `cuida a` un médico. En ocasiones, a las líneas de asociación se les proporcionan dos etiquetas, con el fin de facilitar terminología para leer la asociación en cualquier dirección. Un ejemplo de esto en la Figura 7.10 lo tenemos en la asociación entre las clases `Paciente` y `Habitacion`.

Además de indicar asociaciones entre clases, un diagrama de clases también puede transmitir información acerca de la multiplicidad de esas asociaciones. Es decir, puede indicar cuántas instancias de una clase pueden estar asociadas con instancias de otra. Esta información se anota en los extremos de las líneas de asociación. En particular, la Figura 7.10 indica que cada paciente puede ocupar una habitación y que cada habitación puede alojar a cero o un paciente (suponemos que se trata de habitaciones privadas). Se utiliza un asterisco para indicar un número arbitrario no negativo. Así, el asterisco de la Figura 7.10 indica que cada médico puede cuidar a varios pacientes, mientras que el 1 en el extremo correspondiente al médico significa que a cada paciente solo le cuida un médico (nuestro diseño solo considera el papel de los médicos de atención primaria).

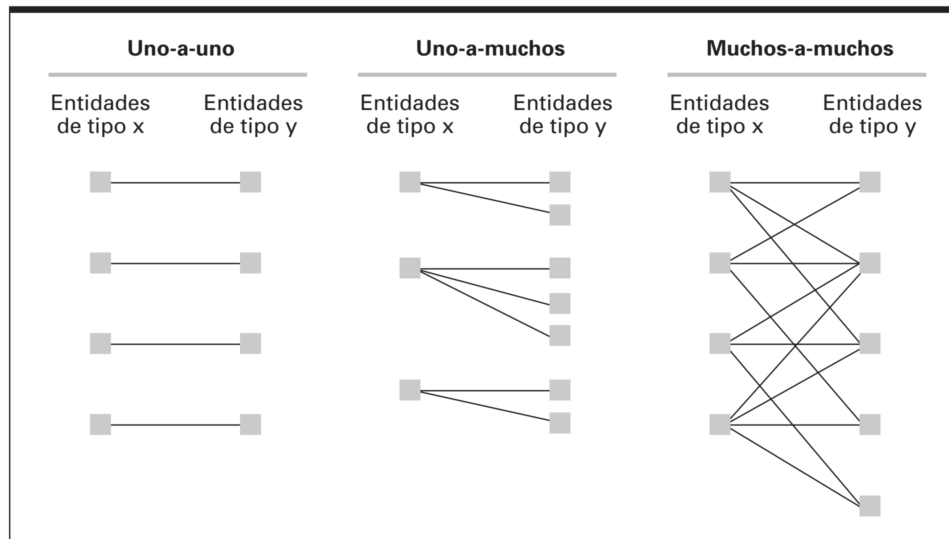
En aras de la exhaustividad, conviene decir que la multiplicidad de las asociaciones se presenta en tres formas básicas: relaciones uno-a-uno, relaciones uno-a-muchos y relaciones muchos-a-muchos, como se resume en la Figura 7.11. Un ejemplo de **relación uno-a-uno** sería la asociación entre pacientes y habitaciones privadas ocupadas en el sentido de que cada paciente está asociado con una única habitación y de que cada habitación ocupada está asociada con un único paciente. Un ejemplo de **relación uno-a-muchos** sería la asociación entre médicos y pacientes, en el sentido de que un médico está asociado con muchos pacientes y de que cada paciente está asociado con solo un médico (de atención primaria). Una **relación muchos-a-muchos** se produciría, por ejemplo, si incluyéramos los médicos especialistas en la relación médico–paciente. Entonces cada médico podría estar asociado con varios pacientes y cada paciente podría estar asociado con varios médicos.

En un diseño orientado a objetos suele darse el caso de que una clase represente una versión más específica de otra. En esas situaciones, decimos que la segunda clase es una generalización de la primera. UML proporciona una notación especial para representar las generalizaciones. En la Figura 7.12 se proporciona un ejemplo en el que se muestran las generalizaciones entre las clases

Figura 7.10 Un diagrama de clases simple.

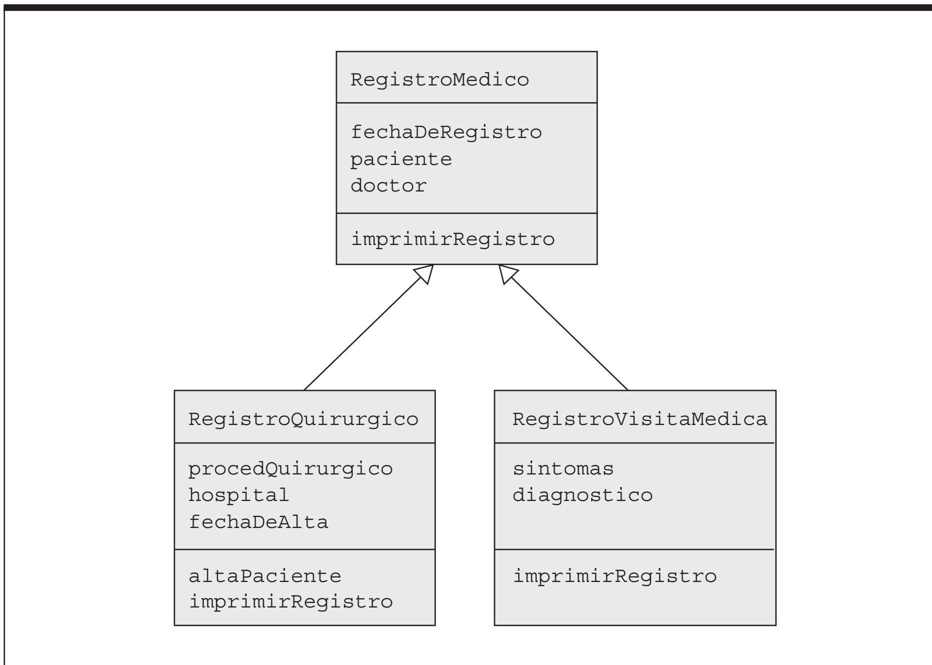


Figura 7.11 Relaciones uno-a-uno, una-a-muchos y muchos-amuchos entre entidades de los tipos X e Y.



`RegistroMedico`, `RegistroQuirurgico` y `RegistroVisitaMedica`. En la figura las asociaciones entre las clases están representadas mediante flechas con la punta hueca, que es la notación que UML utiliza para las asociaciones que son generalizaciones. Observe que cada clase está representada por un rectángulo que contiene el nombre, los atributos y los métodos de la clase en el formato presentado en la Figura 7.4. Esta es la forma que UML tiene de representar las características internas de una clase dentro de un diagrama de clases. La información reflejada en la Figura 7.12 es que la clase `RegistroMedico` es una generalización de la clase `RegistroQuirurgico` y también una generalización de la clase `RegistroVisitaMedica`. Es decir, las clases `RegistroQuirurgico` y `RegistroVisitaMedica` contienen todas las características de la clase `RegistroMedico` más aquellas características explícitamente enumeradas dentro de sus correspondientes rectángulos. Así, tanto las clases `RegistroQuirurgico` como `RegistroVisitaMedica` contienen información del paciente, del médico y de la fecha de registro, pero la clase `RegistroQuirurgico` contiene también información del procedimiento quirúrgico, del hospital, de la fecha de alta, así como la función de dar de alta al paciente, mientras que la clase `RegistroVisitaMedica` contiene información acerca de los síntomas y el diagnóstico. Las tres clases tienen la capacidad de imprimir el registro médico. El método `imprimirRegistro` de `RegistroQuirurgico` y `RegistroVisitaMedica` son especializaciones del método `imprimirRegistro` de `RegistroMedico`; cada uno de esos métodos imprimirá la información específica de su clase.

Figura 7.12 Un diagrama de clases en el que se muestran las generalizaciones.



Recuerde del Capítulo 6 (Sección 6.5) que una forma natural de implementar generalizaciones en un entorno de programación orientado a objetos consiste en utilizar el mecanismo de herencia. Sin embargo, muchos ingenieros de software advierten que la herencia no es apropiada para todos los casos de generalización. La razón es que la herencia introduce un alto grado de acoplamiento entre las clases, un acoplamiento que puede no ser deseable en momentos posteriores del ciclo de vida del software. Por ejemplo, puesto que los cambios dentro de una clase se reflejan automáticamente en todas las clases que heredan de ella, lo que podrían parecer modificaciones de poca entidad durante la etapa de mantenimiento del software pueden tener consecuencias inesperadas. Por ejemplo, suponga que una empresa abriera una instalación recreativa para sus empleados, lo que quiere decir que todas las personas inscritas en esa instalación recreativa serán empleados. Para desarrollar una lista de miembros para estas instalaciones, un programador podría emplear el mecanismo de herencia con el fin de construir una clase `MiembroInstalacionesRecreativas` a partir de una clase `Empleado` previamente definida. Pero si la empresa posteriormente decide abrir las instalaciones recreativas a familiares de los empleados o quizá a antiguos empleados ya jubilados, entonces el acoplamiento predefinido entre la clase `Empleado` y la clase `MiembroInstalacionesRecreativas` tendría que ser deshecho. Por tanto, la herencia no debe utilizarse simplemente por comodidad. En lugar de ello, debe restringirse su uso a aquellos casos en los que la generalización que se esté implementando sea inmutable.

Los diagramas de clases representan características estáticas del diseño de un programa. No representan las secuencias de sucesos que tienen lugar durante la ejecución. Para expresar esas características dinámicas, UML pro-

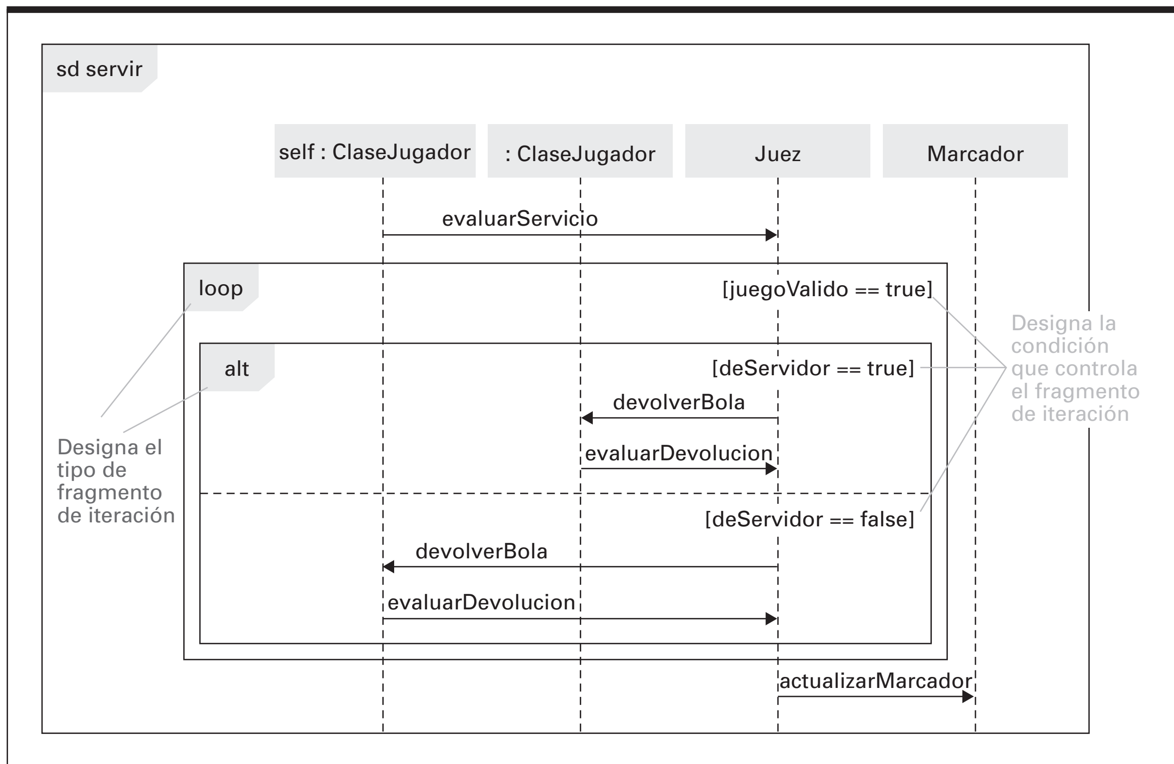
porciona diversos tipos de diagramas que se conocen conjuntamente con el nombre de **diagramas de interacción**. Un tipo de diagrama de interacción es el **diagrama de secuencia** que describe la comunicación entre los individuos (como por ejemplo actores, componentes software completos u objetos individuales) que están involucrados en la realización de una tarea. Estos diagramas son similares al de la Figura 7.5 en el sentido de que representan a los individuos mediante rectángulos con líneas de puntos que salen hacia abajo a partir de ellos. Cada rectángulo junto con su línea de puntos asociada, se denomina **línea de vida**. La comunicación entre los individuos está representada mediante flechas etiquetadas que conectan las líneas de vida apropiadas y en las que la etiqueta indica la acción que se está solicitando. Estas flechas aparecen por orden cronológico si leemos el diagrama de arriba hacia abajo. La comunicación que tiene lugar cuando un individuo completa una tarea solicitada y devuelve el control al individuo solicitante, como en la acción tradicional de volver de un procedimiento, se representa mediante una flecha sin etiquetar que apunta a la línea de vida original.

Así, la Figura 7.5 es esencialmente un diagrama de secuencia. Sin embargo, la sintaxis de la Figura 7.5 tiene por sí sola una serie de limitaciones. Una de ellas es que no nos permite capturar la simetría existente entre los dos jugadores. Debemos dibujar un diagrama separado para representar un tanto que comience con un servicio del `JugadorB`, aún cuando la secuencia de interacción sea muy similar a la que tiene lugar cuando es el `JugadorA` el que sirve. Además, mientras que la Figura 7.5 solo muestra un tanto específico, los tantos en tenis podrían prolongarse indefinidamente. Los diagramas de secuencia formales disponen de técnicas para capturar estas variaciones en un único diagrama y aunque no necesitamos estudiar esas técnicas en detalle, sí que conviene que examinemos brevemente el diagrama de secuencia formal mostrado en la Figura 7.13, que muestra un tanto genérico basado en nuestro diseño de juego de tenis.

Observe también que la Figura 7.13 ilustra que el diagrama de secuencia completo está encerrado en un rectángulo (denominado **marco**). En la esquina superior izquierda del marco aparece un recuadro que contiene los caracteres *sd* (que quiere decir "*sequence diagram*", diagrama de secuencia) seguido por un identificador. Este identificador puede ser un nombre que haga referencia a la secuencia global o, como en el caso de la Figura 7.13, el nombre del método que se invoca para iniciar la secuencia. Observe que, a diferencia de la Figura 7.5, los rectángulos que representan a los jugadores en la Figura 7.13 no hacen referencia a jugadores específicos, sino que simplemente indican que representan objetos del "tipo" `ClaseJugador`. Uno de ellos se etiqueta con la palabra *self,* lo que quiere decir que se trata del jugador cuyo método `servir` está activado para iniciar la secuencia.

Otro punto que hay que resaltar con respecto a la Figura 7.13 tiene que ver con los dos rectángulos internos. Son **fragmentos de interacción**, que se utilizan para representar secuencias alternativas dentro de un diagrama. La Figura 7.13 contiene dos fragmentos de interacción. Uno de ellos está etiquetado como "loop", mientras que el otro está etiquetado como "alt". Esencialmente, son las estructuras while e if-then-else que ya nos hemos encontrado en nuestro pseudocódigo de la Sección 5.2. El fragmento de interacción "loop" indica que los sucesos descritos dentro de sus límites deben repetirse mientras que el objeto

Figura 7.13 Diagrama de secuencia que muestra un tanto genérico.



`Juez` determine que el valor de `juegoValido` sea verdadero (true). El fragmento de interacción "alt" indica que hay que ejecutar una de sus alternativas, dependiendo de si el valor de `deServidor` es verdadero o falso.

Finalmente, aunque no forma parte de UML, es apropiado en este punto hablar del papel de las **tarjetas CRC** (clase-responsabilidad-colaboración) porque desempeñan un papel importante a la hora de validar los diseños orientados a objetos. Una tarjeta CRC es simplemente una tarjeta, como las tarjetas de visita, en las que se escribe la descripción de un objeto. La metodología de las tarjetas CRC consiste en que el diseñador software genere una tarjeta para cada objeto del sistema propuesto y luego las utilice para representar los objetos en una simulación del sistema, quizá sobre una mesa o mediante una simulación "teatral" en la que cada miembro del equipo de diseño sostenga una tarjeta y desempeñe el papel de ese objeto tal y como esa tarjeta lo describe. Tales simulaciones (a menudo denominadas **paseos estructurados**) han resultado ser muy útiles a la hora de identificar errores en un diseño antes de que ese diseño se lleve a implementación.

Patrones de diseño

Una herramienta cada vez más potente para los ingenieros de software es el creciente conjunto de patrones de diseño existente. Un **patrón de diseño** es un modelo predesarrollado para la resolución de un problema recurrente en el diseño software. Por ejemplo, el patrón Adapter (adaptador) proporciona una

solución a un problema que a menudo aparece al construir software a partir de módulos prefabricados. En particular, un módulo prefabricado puede tener la funcionalidad necesaria para resolver el problema que tenemos entre manos, pero puede no disponer de una interfaz compatible con la aplicación actual. En tales casos, el patrón Adapter proporciona una solución estándar para "envolver" dicho módulo dentro de otro módulo que se encargue de realizar la traducción entre la interfaz del módulo original y el mundo exterior, permitiendo así utilizar dentro de la aplicación el módulo original prefabricado.

Otro patrón de diseño bastante consolidado es el patrón Decorator (decorador). Proporciona un medio de diseñar un sistema que realice diferentes combinaciones de las mismas actividades dependiendo de la situación concreta en cada momento. Dichos sistemas pueden llevar a una explosión de opciones que, si no se realiza cuidadosamente el diseño, pueden dar como resultado un software enormemente complejo. Sin embargo, el patrón Decorator proporciona una forma estandarizada de implementar dichos sistemas que conduce a una solución manejable.

La identificación de problemas recurrentes, así como la creación y catalogación de patrones de diseño para resolverlos es un proceso que está actualmente en marcha dentro del campo de la ingeniería del software. Sin embargo, el objetivo no consiste simplemente en encontrar soluciones a los problemas de diseño, sino soluciones de alta calidad que proporcionen flexibilidad en las etapas posteriores del ciclo de vida del software. Así, los principios de diseño fundamentales, como la minimización del acoplamiento y la maximización de la cohesión, desempeñan un papel importante a la hora de desarrollar patrones de diseño.

Los resultados del progreso en el desarrollo de patrones de diseño se ven reflejados en la biblioteca de herramientas proporcionada por los paquetes de desarrollo software actuales, como los entornos de programación ofrecidos por Oracle y el entorno .NET suministrado por Microsoft. De hecho, muchas de las "plantillas" disponibles en estas "cajas de herramientas" son básicamente esqueletos de patrones de diseño que permiten obtener soluciones sencillas y de alta calidad a los problemas de diseño.

Para terminar, debemos mencionar que la aparición de los patrones de diseño en el campo de la ingeniería del software es un ejemplo de cómo los distintos campos del conocimiento pueden contribuir al avance de los demás campos. Los orígenes de los patrones de diseño se remontan a las investigaciones de Christopher Alexander en arquitectura tradicional. Su objetivo era identificar características que contribuyeran a la obtención de diseños arquitectónicos de alta calidad para edificios o complejos de edificios y luego desarrollar patrones de diseño que incorporaran esas características. Actualmente, muchas de sus ideas se han incorporado al diseño software y su trabajo continúa siendo una fuente de inspiración para muchos ingenieros de software.

Cuestiones y ejercicios

1. Dibuje un diagrama de flujo de datos que represente el flujo de datos que se produce cuando un alumno saca en préstamo un libro de una biblioteca.

2. Dibuje un diagrama de casos de uso para un sistema de registros de una biblioteca.
3. Dibuje un diagrama de clases que represente la relación entre los viajeros y los hoteles en los que se hospedan.
4. Dibuje un diagrama de clases que represente el hecho de que una persona es una generalización de un empleado. Indique algunos atributos que puedan pertenecer a cada una de las dos clases.
5. Convierta la Figura 7.5 en un diagrama de secuencia completo.
6. ¿Qué papel desempeñan los patrones de diseño en el proceso de la ingeniería del software?

7.6 Aseguramiento de la calidad

La proliferación de errores en el software, de proyectos cuyo coste se dispara y de fechas de entrega incumplidas exige que se mejoren los métodos de control de la calidad del software. En esta sección vamos a abordar algunas de las direcciones en las que se están llevando a cabo desarrollos dentro de este campo.

El alcance del aseguramiento de la calidad

En los primeros años de las Ciencias de la computación, el problema de generar software de calidad se centraba en eliminar los errores de programación que se hubieran cometido durante la implementación. Posteriormente en esta sección veremos qué progresos se han hecho en este sentido. Sin embargo, hoy día, el alcance del control de la calidad del software va mucho más allá del proceso de depuración, con ramificaciones que incluyen la mejora de los procedimientos de la ingeniería del software, el desarrollo de programas de formación, que en muchos casos conducen a un proceso de certificación formal, y el establecimiento de estándares en los que poder basar una adecuada ingeniería del software. A este respecto, ya hemos resaltado el papel de organizaciones tales como ISO, IEEE y ACM a la hora de mejorar la profesionalidad y de establecer estándares para valorar el control de calidad existente dentro de las empresas de desarrollo software. Un ejemplo específico es la serie ISO 9000 de estándares, que abarca numerosas actividades industriales, tales como el diseño, la producción, la instalación y el servicio post-venta. Otro ejemplo es el conjunto de estándares ISO/IEC 15504 desarrollado conjuntamente por ISO y la Comisión Internacional Electrotécnica (IEC, *International Electrotechnical Commission*).

Muchos de los principales contratistas de software exigen ahora que las organizaciones a las que subcontratan para desarrollar software se ajusten a esos estándares. Como resultado, las empresas de desarrollo software están estableciendo **grupos de aseguramiento de la calidad del software** (SQA, *Software Quality Assurance*), que están a cargo de vigilar e imponer los sistemas de control de calidad adoptados por la organización. Así, en el caso del modelo tradicional en cascada, el grupo de aseguramiento de la calidad del software estaría encargado de la tarea de aprobar la especificación de requisitos software antes de que dé comienzo la etapa de diseño o de aprobar el diseño y los documentos relacionados antes de iniciar la implementación.

Tragedias históricas del diseño de sistemas

Un buen ejemplo de la necesidad de disciplinas de diseño apropiadas son los problemas con los que se encontró el Therac-25, que era un sistema de radioterapia acelerador de electrones basado en computadora, que la comunidad médica estuvo utilizando a mediados de la década de 1980. Los fallos en el diseño de la máquina contribuyeron a la aparicion de seis casos de sobredosis de radiación, tres de ellos con resultado de muerte. Entre los fallos se incluían (1) un diseño inadecuado de la interfaz de la máquina, que permitía que el operador comenzara a aplicar la radiación antes de que la máquina se hubiera ajustado para la dosis apropiada y (2) una mala coordinación entre el diseño del hardware y del software, que condujo a que no se incorporaran ciertas características de seguridad.

En otros casos más recientes, el inadecuado diseño ha conducido a apagones generalizados, interrupciones del servicio telefónico, errores importantes en transacciones financieras, pérdida de sondas espaciales e interrupciones de las comunicaciones por Internet. Puede conocer más datos acerca de este tipo de problemas en el Risks Forum que puede encontrar en la dirección `http://catless.ncl.ac.uk/Risks`.

Son varios los temas que subyacen a los esfuerzos actuales que se realizan para llevar a cabo el control de calidad. Uno de ellos es el mantenimiento de los registros adecuados. Es crucial que cada paso del proceso de desarrollo se documente con precisión para futura referencia. Sin embargo, este objetivo entra en conflicto con la propia naturaleza humana. Todos sentimos alguna vez la tentación de tomar decisiones o modificar decisiones sin actualizar los documentos relacionados. El resultado es que hay una probabilidad de que los registros sean incorrectos y de que, por tanto, su uso en etapas futuras provoque confusión. Aquí radica precisamente una de las mayores ventajas de las herramientas CASE. Esas herramientas hacen que realizar determinadas tareas sea mucho más fácil que con los métodos manuales, como por ejemplo redibujar diagramas y actualizar diccionarios de datos. En consecuencia, es más probable que las actualizaciones se lleven a cabo y que la documentación final tenga la precisión suficiente. (Este ejemplo es solo uno de los muchos casos en los que la ingeniería del software debe lidiar con los defectos inherentes a la naturaleza humana. Otros ejemplos incluyen los inevitables conflictos de personalidades, los celos y las peleas de egos que surgen cuando las personas trabajan juntas.).

Otro de los asuntos fundamentales relacionados con la calidad es el uso de **revisiones**, en las que varias de las partes implicadas en un proyecto de desarrollo software se reúnen para tener en cuenta un tema específico. Las revisiones se producen a todo lo largo del proceso de desarrollo del software, adoptando la forma de revisiones de requisitos, revisiones de diseño y revisiones de implementación. Pueden consistir en demostraciones de los prototipos en las primeras etapas del análisis de requisitos, en paseos estructurados realizados por los miembros del equipo de diseño software o simplemente en reuniones de coordinación de programadores que estén implementando partes relacionadas del diseño. Estas revisiones, realizadas de forma recurrente, proporcionan canales de comunicación con los que evitar los malos entendidos y con los que corregir los errores antes de que se transformen en desastres. La

importancia de las revisiones queda ilustrada por el hecho de que están contempladas específicamente en el estándar de Revisiones Software del IEEE, conocido como IEEE 1028.

Algunas revisiones son absolutamente cruciales. Un ejemplo sería la reunión de revisión celebrada por los representantes de las partes interesadas en un proyecto y el equipo de desarrollo software, en la que se aprueba la especificación final de requisitos software. De hecho, esta aprobación marca el final de la fase de análisis de requisitos formal y es la base a partir de la cual progresarán las restantes etapas de desarrollo. Sin embargo, todas las revisiones son importantes y en aras del control de calidad, hay que documentarlas, como parte del proceso continuado del mantenimiento de la documentación del desarrollo.

Pruebas del software

Aunque ahora el aseguramiento de la calidad del software se considera un tema que impregna todo el proceso de desarrollo, las pruebas y la verificación de los propios programas continúan siendo tema de investigación. En la Sección 5.6 hemos hablado de las técnicas para verificar la corrección de los algoritmos de una forma matemáticamente rigurosa y allí concluimos que la mayoría del software actual se "verifica" por medio de pruebas. Lamentablemente, tales pruebas son, en el mejor de los casos, inexactas. No es posible garantizar mediante pruebas que un programa software es correcto a menos que ejecutemos las pruebas suficientes como para abarcar todos los posibles escenarios. Pero incluso en los programas más simples puede haber miles de millones de rutas diferentes que pueden potencialmente recorrerse. Por tanto, probar todas las posibles rutas dentro de un programa complejo es una tarea imposible.

Por otro lado, los ingenieros de software han desarrollado metodologías de pruebas que mejoran la probabilidad de detectar errores en el software utilizando un número limitado de pruebas. Una de estas técnicas se basa en la observación de que los errores en el software tienden a agruparse. Es decir, la experiencia demuestra que un pequeño número de módulos dentro de un sistema software de gran tamaño tiende a ser más problemático que el resto. Así, identificando esos módulos y probándolos más exhaustivamente, pueden descubrirse más errores del sistema que si todos los módulos se probaran de una forma uniforme y menos exhaustiva. Este es un ejemplo del principio conocido con el nombre de **principio de Pareto**, en referencia al economista y sociólogo Vilfredo Pareto (1848–1923) que observó que una pequeña parte de la población italiana controlaba la mayor parte de la riqueza del país. En el campo de la ingeniería del software, el principio de Pareto afirma que los resultados pueden a menudo mejorarse de la manera más rápida concentrando los esfuerzos en un área determinada.

Otra metodología de prueba del software, denominada **prueba del camino básico**, consiste en desarrollar un conjunto de datos de prueba que garantice que cada instrucción del software se ejecute al menos una vez. Se han desarrollado técnicas para identificar esos conjuntos de datos de prueba utilizando un área de las matemáticas conocida como teoría de grafos. Así, aunque pueda ser imposible garantizar que se prueben todos los caminos a través de un sistema

software, es posible asegurar que todas las sentencias del sistema se ejecutan al menos una vez durante el proceso de pruebas.

Las técnicas basadas en el principio de Pareto y en las pruebas del camino básico dependen del conocimiento de la composición interna del software que se esté probando. Caen, por tanto, dentro de la categoría denominada **pruebas de caja de cristal**, que quiere decir que la persona a cargo de las pruebas de software es consciente de la estructura interior del software y utiliza ese conocimiento a la hora de diseñar las pruebas. Por contraste, existe otra categoría de pruebas, denominada **pruebas de caja negra**, que hace referencia a aquellas pruebas que no dependen del conocimiento de la composición interna del software. En pocas palabras, las pruebas de caja negra se realizan desde el punto de vista del usuario. En las pruebas de caja negra no nos preocupa la forma en que el software lleva a cabo su tarea, sino simplemente si realiza esa tarea correctamente en términos de precisión y velocidad.

Un ejemplo de pruebas de caja negra sería la técnica denominada **análisis de valores límite**, que consiste en identificar rangos de datos, denominados **clases de equivalencia**, para los que el software debería comportarse de forma similar y luego probar el software con datos próximos a los límites de dichos rangos. Por ejemplo, si se supone que el software debe aceptar valores de entrada dentro de un rango especificado, entonces podríamos probar el software con los valores más bajos y más altos del rango; o bien, si se supone que el software tiene que coordinar varias actividades, entonces podríamos probarlo empleando el mayor conjunto de actividades posible. La teoría subyacente es que al identificar las clases de equivalencia, podemos minimizar el número de casos de prueba, porque una correcta operación para unos cuantos ejemplos de una clase de equivalencia tiende a validar el software para toda esa clase. Además, la mayor probabilidad de identificar un error dentro de una clase consiste en utilizar los datos correspondientes a los límites de esa clase.

Otra metodología que cae dentro de la categoría de pruebas de caja negra son las denominadas **pruebas beta**, en las que se proporciona una versión preliminar del software a un segmento del público objetivo con el fin de comprobar cómo se comporta el software en situaciones de la vida real, antes de solidificar la versión final del producto y lanzarla al mercado. (Las pruebas similares realizadas en las instalaciones del desarrollador se denominan **pruebas alfa**.) Las ventajas de las pruebas beta van mucho más allá de la tarea tradicional de descubrimiento de errores. Con esas pruebas, se obtiene una realimentación de carácter general por parte de los clientes (tanto positiva como negativa) que puede ser de ayuda a la hora de perfeccionar las estrategias de mercado. Además, una distribución temprana del software en versión beta ayuda a otros desarrolladores de software a diseñar productos compatibles con el nuestro. Por ejemplo, en el caso de un nuevo sistema operativo para el mercado del PC, la distribución de una versión beta anima al desarrollo de utilidades compatibles, de forma que el sistema operativo final termine por aparecer en las estanterías de los comercios rodeado por otros productos de acompañamiento. Además, la realización de pruebas beta puede generar una sensación de anticipación en el mercado, una atmósfera que incrementa la publicidad y las ventas.

Cuestiones y ejercicios

1. ¿Qué papel desempeña el grupo SQA dentro de una organización de desarrollo de software?
2. ¿De qué maneras tiende la naturaleza humana a obstaculizar el aseguramiento de la calidad?
3. Identifique dos temas que se aplican a lo largo de todo el proceso de desarrollo para mejorar la calidad.
4. A la hora de probar un software, ¿cuándo decimos que una prueba ha tenido éxito, cuando encuentra errores o cuando no los encuentra?
5. ¿Qué técnicas propondría para identificar los módulos de un sistema que deben ser sometidos a pruebas más exhaustivas que los restantes?
6. ¿Qué prueba se podría realizar en un paquete software que hubiera sido diseñado para ordenar una lista de no más de 100 entradas?

7.7 Documentación

Un sistema software tiene poca utilidad a menos que las personas puedan aprender a utilizarlo y a mantenerlo. Por tanto, la documentación es una parte importante del paquete software final y como consecuencia uno de los temas importantes de la ingeniería del software.

La documentación del software sirve para tres cosas distintas, lo que conduce a que existan tres categorías de documentación: documentación del usuario, documentación del sistema y documentación técnica. El propósito de la **documentación del usuario** es explicar las características del software y describir cómo utilizarlo. Está pensada para ser leída por el usuario del software y se expresa por tanto utilizando la terminología de la aplicación.

Hoy día, la documentación del usuario se reconoce como una herramienta de marketing importante. Una buena documentación del usuario combinada con una interfaz de usuario bien diseñada hace que un paquete software sea más accesible e incrementa, por tanto, las ventas. Siendo conscientes de esto, muchos desarrolladores de software contratan a escritores de documentación técnica para generar esta parte del producto, o bien proporcionan versiones preliminares de sus productos a autores independientes, para que haya disponibles libros prácticos en las librerías en el momento de presentar el software al público.

La documentación de usuario suele adoptar la forma de un libro o de un folleto físicos, pero en muchos casos la misma información se incluye como parte del propio software. Esto permite al usuario consultar la documentación mientras utiliza el software. En este caso, la información puede descomponerse en pequeñas unidades, que en ocasiones se denominan paquetes de ayuda, que pueden aparecer automáticamente en la pantalla si el usuario tarda demasiado tiempo entre un comando y otro.

El propósito de la **documentación del sistema** es describir la composición interna del software, de modo que este pueda ser mantenido en las etapas posteriores de su ciclo de vida. Un componente principal de la documentación del

sistema es la versión fuente de todos los programas que componen el sistema. Es importante que estos programas se presenten en un formato legible, razón por la cual los ingenieros de software apoyan el uso de lenguajes de programación de alto nivel bien diseñados, el uso de sentencias de comentario para explicar un programa y un diseño modular que permita presentar cada módulo como una unidad coherente. De hecho, la mayoría de las empresas que fabrican productos software han adoptado una serie de convenios que sus empleados deben seguir a la hora de escribir los programas. Estos convenios incluyen normas de sangrado para organizar un programa en la página escrita, convenios de denominación que establecen una distinción entre los diversos elementos de un programa, como variables, constantes, objetos y clases; y convenios de documentación que sirven para garantizar que todos los programas estén suficientemente documentados. Tales convenios permiten conseguir un cierto grado de uniformidad en todo el software de una empresa, lo que termina por simplificar el proceso de mantenimiento del software.

Otro componente de la documentación del sistema es un registro de los documentos de diseño, incluyendo la especificación de requisitos del software y la forma en la que estas especificaciones se obtuvieron durante el diseño. Esta información resulta útil durante el mantenimiento del software porque indica la razón por la que el software fue implementado de esa forma concreta; además, esta información reduce la posibilidad de que los cambios realizados durante el mantenimiento destruyan la integridad del sistema.

El propósito de la **documentación técnica** es describir cómo hay que instalar y dar servicio a un sistema software (como por ejemplo cuál es la forma de ajustar los parámetros de operación, cómo instalar actualizaciones y cómo informar de los problemas al desarrollador del software). La documentación técnica del software es análoga a la documentación proporcionada a los mecánicos en el sector del automóvil. Esta documentación no entra a analizar cómo se diseñó y se construyó el vehículo (que sería lo análogo a la documentación del sistema), ni tampoco explica cómo conducir el vehículo o cómo hacer funcionar el sistema de calefacción/aire acondicionado (lo que sería análogo a la documentación del usuario). En lugar de ello, describe cómo reparar o mantener los componentes del vehículo, por ejemplo cómo sustituir la transmisión o cómo localizar un problema eléctrico intermitente.

La diferencia entre la documentación técnica y la documentación del usuario se difumina en el campo del PC, porque a menudo es el propio usuario el que se encarga también de mantener e instalar el software. Sin embargo, en los entornos multiusuario, esa distinción es más clara. Por ello, la documentación técnica está dirigida al administrador del sistema, que es responsable de dar servicio a todo el software que cae bajo su jurisdicción, permitiendo a los usuarios acceder a los paquetes software como si fueran herramientas abstractas.

Cuestiones y ejercicios

1. ¿De qué maneras puede documentarse el software?
2. ¿En qué fase (o fases) del ciclo de vida del software se prepara la documentación del sistema?
3. ¿Qué es más importante, un programa o su documentación?

7.8 La interfaz persona-máquina

Recuerde de la Sección 7.2, que una de las tareas durante el análisis de requisitos es definir cómo interactuará el sistema software propuesto con su entorno. En esta sección vamos a dirigir nuestra atención hacia los temas asociados con esta interacción, en aquellos casos en que implica comunicarse con los seres humanos, un tema con profundas implicaciones. Después de todo, los seres humanos deberían poder emplear un sistema software como una herramienta abstracta. Esta herramienta debería ser fácil de aplicar y estar diseñada para minimizar (idealmente eliminar) los errores de comunicación entre el sistema y los usuarios humanos. Esto significa que la interfaz del sistema debe diseñarse para la comodidad de las personas, en lugar de para simplificar el sistema software.

La importancia de un buen diseño de interfaz se ve más reforzada aún por el hecho de que es la interfaz del sistema, no ninguna de sus otras características, la que probablemente vaya a causar una mayor impresión en el usuario. Después de todo, los seres humanos tendemos a ver un sistema en términos de su usabilidad, no en función de lo inteligentemente que realice sus tareas internas. Desde la perspectiva de un ser humano, la elección entre dos sistemas competidores se basará probablemente en las interfaces de ambos sistemas. Por tanto, el diseño de la interfaz de un sistema puede llegar a ser el factor determinante del éxito o el fracaso de un proyecto de ingeniería del software.

Por estas razones, la interfaz persona-máquina se ha convertido en una de las preocupaciones principales durante la etapa de definición de requisitos de los proyectos de desarrollo software y es uno de los campos de mayor crecimiento dentro de la ingeniería del software. De hecho, algunos expertos argumentan que el estudio de las interfaces persona-máquina es un campo independiente por propio derecho.

Uno de los beneficiarios de la investigación en este campo es la interfaz de los teléfonos inteligentes. Para conseguir el objetivo de obtener un dispositivo de bolsillo suficientemente cómodo de utilizar, se están sustituyendo los elementos de la interfaz persona-máquina tradicionales (teclado completo, ratón, barras de desplazamiento, menús) por nuevas soluciones, como por ejemplo gestos realizados en una pantalla táctil, comandos de voz y teclados virtuales con autoterminación inteligente de palabras y frases. Aunque todo esto representa un progreso bastante significativo, la mayor parte de los usuarios de teléfonos inteligentes opinarían que existe todavía un amplio margen para ulteriores innovaciones.

La investigación en el diseño de interfaces persona-máquina se apoya fuertemente en las áreas de la ingeniería denominadas **ergonomía**, que trata con el diseño de sistemas que armonicen con las capacidades físicas de los seres humanos, y **cognética**, que trata del diseño de sistemas que armonicen con las habilidades mentales de las personas. De las dos, la ergonomía es la que mejor se comprende, fundamentalmente porque los seres humanos llevamos siglos interaccionando físicamente con máquinas. Podemos encontrar ejemplos en las antiguas herramientas, en las armas utilizadas a lo largo de la historia y en los sistemas de transporte. Buena parte de las lecciones en este campo son evidentes por sí mismas, a pesar de lo cual ha habido ocasiones en las que la aplicación de la ergonomía ha sido de lo más anti-intuitiva. Un ejemplo muy citado

es el diseño del teclado de la máquina de escribir (ahora reencarnado en los teclados de las computadoras) en los que las teclas se dispusieron de manera intencionada para reducir la velocidad del mecanógrafo, con el fin de que no se atascaran los sistemas mecánicos de palancas utilizados en las primeras máquinas.

La interacción mental con las máquinas, por el contrario, es un fenómeno relativamente nuevo y es por eso que la cognética ofrece el mayor potencial para futuras investigaciones de provecho y para el aprendizaje de nuevas lecciones acerca de la interacción persona-máquina. A menudo, los resultados que se obtienen son interesantes por su sutileza. Por ejemplo, los seres humanos tendemos a desarrollar hábitos, una costumbre que, analizada superficialmente, parece buena porque puede aumentar la eficiencia. Pero los hábitos pueden conducir también a errores, aún cuando el diseño de una interfaz contemple de manera intencionada el problema. Considere por ejemplo el proceso por el que un ser humano solicita a un sistema operativo típico que borre un archivo. Para evitar borrados no intencionados, la mayoría de las interfaces responden a una de esas solicitudes pidiendo al usuario que confirme la solicitud, quizá mediante un mensaje del tipo "¿Quiere realmente borrar este archivo?". A primera vista, esta petición de confirmación parece resolver cualquier posible problema con los borrados no intencionados. Sin embargo, después de utilizar el sistema durante un periodo prolongado, el ser humano desarrolla el hábito de responder automáticamente a la cuestión con un "sí". De este modo, la tarea de borrar archivos deja de ser un proceso de dos pasos formado por un comando de borrado seguido de una meditada respuesta a una cuestión. En lugar de ello, se convierte en un proceso de un único paso, de tipo "borrar-sí", lo que implica que para cuando la persona se da cuenta de que ha efectuado una solicitud incorrecta de borrado, esa solicitud ya ha sido confirmada y el borrado ya se ha producido.

La formación de hábitos también puede causar problemas cuando se le pide a un ser humano que emplee varios paquetes de software de aplicación. Las interfaces de dichos paquetes pueden ser similares, pero con pequeñas diferencias. De ese modo, acciones similares del usuario pueden dar como resultado respuestas del sistema diferentes; a la inversa, respuestas del sistema similares pueden requerir diferentes acciones del usuario. En estos casos, los hábitos desarrollados con una aplicación pueden llevar a errores en las otras.

Otra característica humana que preocupa a los investigadores del campo del diseño de las interfaces persona-máquina es lo estrecho que es el campo de atención de las personas, que tiende a estrecharse cada vez más a medida que el nivel de concentración se incrementa. Cuando un ser humano se sumerge cada vez más en la tarea que tiene entre manos, romper esa concentración puede ser difícil. En 1972, un avión comercial se estrelló porque los pilotos estaban tan absortos con un problema del tren de aterrizaje (de hecho, con el proceso de cambiar la señal luminosa del indicador del tren de aterrizaje) que permitieron que el avión chocara contra el suelo, a pesar de que las alarmas estaban sonando dentro de la cabina.

En las interfaces de PC aparecen continuamente otros ejemplos menos críticos. Por ejemplo, la mayoría de los teclados disponen de un indicador luminoso "Bloq Mayus" para señalar que el teclado está en modo de escritura en letras mayúsculas (es decir, que la tecla "Bloq Mayus" ha sido pulsada). Sin

embargo, si se pulsa la tecla por accidente, los seres humanos rara vez nos percataremos del estado del indicador luminoso hasta que veamos aparecer caracteres extraños en la pantalla. Incluso entonces, el usuario se siente sorprendido durante unos instantes por ese hecho, hasta darse cuenta de cuál es la causa del problema. En un cierto sentido, no es sorprendente, el indicador luminoso del teclado no está en el campo de visión del usuario. Sin embargo, los usuarios tienden a menudo a no percibir los indicadores colocados directamente en su línea de visión. Por ejemplo, los usuarios pueden estar tan absortos en una tarea que no noten los cambios en la apariencia del cursor de pantalla, aún cuando su tarea implique estar observando ese cursor.

Otra característica de los seres humanos que es preciso tener en cuenta durante el diseño de interfaces es la capacidad limitada de nuestra mente para tratar con varios hechos de manera simultánea. En un artículo publicado en la revista ***Psychological Review,*** en 1956, George A. Miller informó de sus investigaciones que indicaban que la mente humana solo es capaz de tratar con unos siete detalles al mismo tiempo. Por ello, es importante que una interfaz se diseñe para presentar toda la información relevante en el momento que es necesario tomar una decisión, en lugar de confiar en la memoria del usuario de la máquina. En particular, no sería muy adecuado como diseño exigir a una persona que se acuerde de detalles muy precisos mostrados anteriormente en la pantalla. Más aún, si una interfaz requiere desplazarse frecuentemente entre unas pantallas y otras, los seres humanos pueden perderse rápidamente en el laberinto. Por ello, el contenido y la disposición de las imágenes en pantalla es uno de los aspectos más importantes de diseño.

Aunque la aplicación de la ergonomía y la cognética proporcionan al campo del diseño de interfaces persona-máquina unas características distintivas, ese campo también abarca muchos de los temas más tradicionales de la ingeniería del software. En particular, la búsqueda de métricas es tan importante en el campo del diseño de interfaces como en las áreas más tradicionales de la ingeniería del software. Entre las características de las interfaces que se han sometido a medida están el tiempo requerido para aprender a usar una interfaz, el tiempo requerido para realizar determinadas tareas con esa interfaz, la tasa de errores de la interfaz de usuario, el grado con el que el usuario retiene su dominio sobre la interfaz después de estar un cierto periodo sin utilizarla e incluso algunos aspectos tan subjetivos como el grado con el que la interfaz gusta a los usuarios.

El modelo **GOMS** (***Goals, Operators, Methods and Selection rules***), presentado originalmente en 1954, es representativo de la búsqueda de métricas en el campo del diseño de interfaces persona-máquina. La metodología subyacente en este modelo consiste en analizar tareas en términos de objetivos del usuario (como por ejemplo borrar una palabra de un texto), de los operadores (como por ejemplo hacer clic con el botón de un ratón), de los métodos (como hacer doble clic con el botón del ratón y pulsar la tecla de borrado) y de las reglas de selección (como por ejemplo elegir entre dos métodos de conseguir un mismo objetivo). Este es, de hecho, el origen del acrónimo GOMS (***Goals, Operators, Methods and Selection rules***; objetivos, operadores, métodos y reglas de selección). En pocas palabras, GOMS es una metodología que permite analizar las acciones de una persona que esté utilizando una interfaz en forma de secuencias de pasos elementales (pulsar una tecla, mover el ratón, tomar una deci-

sión). Al rendimiento de cada paso elemental se le asigna un periodo de tiempo preciso y así, sumando los tiempos asignados a los distintos pasos que componen una tarea, GOMS proporciona un medio de comparar diferentes interfaces propuestas en función del tiempo que cada una de ellas requiere a la hora de realizar tareas similares.

El propósito de nuestro estudio no es comprender los detalles de sistemas tales como GOMS. Lo que queremos resaltar es que GOMS se basa en características del comportamiento humano (mover las manos, tomar decisiones, etc.). De hecho, el desarrollo de GOMS se consideró originalmente un tema del campo de la psicología. Por eso, GOMS recalca el papel que las características de la naturaleza humana desempeñan en el campo del diseño de interfaces persona-máquina, incluso en aquellos temas que se han heredado de la ingeniería del software tradicional.

El diseño de interfaces persona-máquina promete ser un campo de investigación activo en los próximos años. Muchos de los problemas relacionados con las interfaces gráficas de usuario actuales están todavía por resolver, y ya es posible intuir una multitud de problemas adicionales relativos al uso de las interfaces tridimensionales que se vislumbran en el horizonte. De hecho, puesto que estas interfaces prometen combinar las comunicaciones audio y táctiles con la visión tridimensional, la gama de posibles problemas es enorme.

Cuestiones y ejercicios

1. **a.** Identifique una aplicación de la ergonomía en el campo del diseño de interfaces persona-computadora.
   b. Identifique una aplicación de la cognética en el campo del diseño de interfaces persona-computadora.
2. Una diferencia notable en la interfaz persona-computadora de un teléfono inteligente con respecto a la de una computadora de sobremesa son las técnicas utilizadas para desplazar una parte de la pantalla. En una computadora de sobremesa, el desplazamiento suele conseguirse arrastrando el ratón sobre las barras de desplazamiento mostradas en el lado derecho y el lado inferior de la región que se quiere desplazar. Por el contrario, las barras de desplazamiento no suelen utilizarse en un teléfono inteligente (si se utilizan, aparecen como líneas finas para indicar qué parte de la imagen subyacente es actualmente visible). El desplazamiento se efectúa por tanto mediante el gesto de un toque deslizante sobre la pantalla.
   a. Basándose en la ergonomía, ¿qué argumentos pueden darse para defender esta diferencia?
   b. Basándose en la cognética, ¿qué argumentos pueden darse para defender esta diferencia?
3. ¿Qué distingue el campo del diseño de interfaces persona-máquina del campo más tradicional de la ingeniería del software?
4. Identifique tres características humanas que deberían ser tenidas en cuenta a la hora de diseñar una interfaz persona-máquina.

7.9 Propiedad del software y responsabilidad legal

La mayoría de las personas estarían de acuerdo en que una empresa o un individuo tienen derecho a recuperar la inversión necesaria para desarrollar un software de calidad y a obtener beneficio económico de la misma. En caso contrario, es poco probable que hubiera muchas personas dispuestas a acometer la tarea de fabricar el software que nuestra sociedad desea. En pocas palabras, los desarrolladores de software necesitan que se les reconozca una cierta propiedad sobre el software que producen.

Los esfuerzos legales para defender esa propiedad caen bajo la categoría de las leyes de la **propiedad intelectual**, buena parte de las cuales están basadas en los principios bien establecidos del copyright y las leyes de patentes. De hecho, el propósito de un copyright o de una patente es permitir al desarrollador de un "producto" comercializar ese producto (o partes del mismo) a las personas interesadas, al mismo tiempo que se protegen sus derechos de propiedad. Por ello, el desarrollador de un producto (ya sea un individuo o una empresa) dejará claros sus derechos de propiedad incluyendo una declaración de copyright en todas las obras producidas; esto incluye las especificaciones de requisitos, los documentos de diseño, el código fuente, los planes de pruebas, además de colocar ese aviso de copyright en algún lugar visible del producto final. El aviso de copyright identifica claramente la propiedad, el personal autorizado para usar ese trabajo y otras restricciones. Además, los derechos del desarrollador se expresan formalmente en términos legales en una **licencia de software**.

Una licencia de software es un acuerdo legal entre el propietario y el usuario de un producto software que concede al usuario ciertos permisos para utilizar el producto sin transferirle los derechos de propiedad intelectual sobre el mismo. Estos acuerdos indican, con un alto grado de detalle, los derechos y obligaciones de ambas partes. Por tanto, es importante leer cuidadosamente y comprender los términos de la licencia de software antes de instalar y utilizar un producto software.

Aunque el copyright y los acuerdos de licencia de software proporcionan medios legales para impedir la copia directa y el uso no autorizado del software, suelen ser insuficientes a la hora de impedir que algún otro individuo o empresa desarrolle independientemente un producto con una función prácticamente idéntica. Es triste que a lo largo de los años haya habido tantas ocasiones en las que el desarrollador de un producto software verdaderamente revolucionario fuera incapaz de beneficiarse de su invento (dos ejemplos notables son las hojas de cálculo y los navegadores web). En la mayor parte de estos casos, surgió otra empresa que tuvo éxito a la hora de desarrollar un producto competidor que se aseguró una posición dominante en el mercado. La ley de patentes ofrece un mecanismo legal para impedir este tipo de intrusiones por parte de los competidores.

Las leyes de patentes se establecieron para permitir que un inventor se beneficie comercialmente de su invento. Para obtener una patente, el inventor debe proporcionar los detalles de su invento y demostrar que se trata de algo nuevo, útil y que no resulta obvio para otras personas con una preparación similar (un requisito que puede ser bastante complicado en el caso del software). Si se concede una patente, al inventor se le da el derecho de impedir que

otros fabriquen, utilicen, vendan o importen ese invento durante un periodo limitado de tiempo, que normalmente es de veinte años a partir del momento que se registró la solicitud de patente.

Un inconveniente del uso de patentes es que el proceso de obtener una patente es muy caro y lleva mucho tiempo, requiriendo a menudo varios años. Durante ese tiempo, un producto software puede llegar a quedarse obsoleto; además, hasta que se concede la patente, la autoridad del solicitante para impedir que otros se apropien del producto es cuestionable.

La importancia de reconocer el copyright, las licencias de software y las patentes es enorme en el proceso de la ingeniería del software. A la hora de desarrollar un producto software, los ingenieros de software a menudo incorporan software de otros productos; ya sea un producto completo, un subconjunto de componentes o incluso partes de código fuente descargadas a través de Internet. Sin embargo, el no reconocer los derechos de propiedad intelectual durante este proceso puede conducir a responsabilidades legales y tener graves consecuencias. Por ejemplo, en 2004, una pequeña empresa poco conocida, NPT Inc., ganó una demanda contra Research In Motion (RIM, los fabricantes de los teléfonos inteligentes BlackBerry) por infracción de patente con respecto a unas cuantas tecnologías clave incorporadas en los sistemas de correo electrónico de RIM. ¡La sentencia incluía la orden de suspender los servicios de correo electrónico a todos los usuarios de BlackBerry en Estados Unidos! RIM terminó por llegar a un acuerdo para pagar a NPT un total de 612,5 millones de dólares, evitando así la interrupción del servicio.

Por último, debemos dedicar unas palabras al tema de las responsabilidades legales. Para protegerse a sí mismos frente a esas responsabilidades, los desarrolladores de software suelen incluir en las licencias de software declaraciones de renuncia de responsabilidad que indican las limitaciones de sus responsabilidades. En esas declaraciones son comunes frases como "En ningún caso, la empresa X será responsable de los daños que se produzcan como resultado del uso de este software". Sin embargo, los tribunales raramente admiten una de esas renuncias de responsabilidad si el demandante puede demostrar la negligencia por parte del demandado. Por ello, los casos de responsabilidad civil tienden a centrarse en si el demandado utilizó un grado de cuidado compatible con el producto que se estaba desarrollando. Un grado de cuidado que pueda considerarse aceptable en el caso de estar desarrollando un sistema de procesamiento de textos puede considerarse negligente si se está desarrollando software para controlar un reactor nuclear. En consecuencia, una de las mejores defensas contra las posibles demandas por el uso del software consiste en aplicar principios adecuados de ingeniería del software durante el desarrollo, utilizar un nivel de cuidado compatible con la aplicación que se le vaya a dar al software y generar y mantener registros que permitan demostrar los esfuerzos realizados.

Cuestiones y ejercicios

1. ¿Qué importancia tienen los avisos de copyright en las especificaciones de requisitos, los documentos de diseño, el código fuente y el producto final?

2. ¿En qué sentido benefician a la sociedad las leyes de propiedad intelectual y de patentes ?
3. ¿En qué casos no admiten los tribunales las declaraciones de renuncia de responsabilidad?

Problemas de repaso

(Los problemas con asterisco están asociados con las secciones opcionales.)

1. Proporcione un ejemplo de cómo los esfuerzos realizados durante el desarrollo del software pueden resultar ventajosos posteriormente en el mantenimiento de ese software.
2. ¿Qué es el prototipado?
3. Explique cómo el uso de métricas para medir ciertas propiedades del software afecta a la ingeniería del software.
4. ¿Cabría esperar que una métrica para medir la complejidad de un sistema software fuera acumulativa en el sentido de que la complejidad de un sistema completo fuese la suma de las complejidades de sus partes? Explique su respuesta.
5. ¿Cabría esperar que una métrica para medir la complejidad de un sistema software fuera conmutativa en el sentido de que la complejidad de un sistema completo fuese la misma si se desarrollara originalmente con la funcionalidad X y posteriormente se le añadiera la funcionalidad Y, o si se desarrollara originalmente con la funcionalidad Y y se le añadiera después la funcionalidad X? Explique su respuesta.
6. ¿En qué difiere la ingeniería del software de otros campos más tradicionales de la ingeniería, como por ejemplo la eléctrica o la mecánica?
7. a. Identifique una desventaja del modelo en cascada tradicional para el desarrollo de software.
   b. Identifique una ventaja del modelo en cascada tradicional para el desarrollo de software.
8. ¿Qué tipo de metodología es el desarrollo de código fuente abierto, de tipo arriba-abajo o del tipo abajo-arriba? Explique su respuesta.
9. Indique por qué el uso de constantes en lugar de literales puede simplificar el mantenimiento del software.
10. ¿Cuál es la diferencia entre el acoplamiento y la cohesión? ¿Cuál hay que minimizar y cuál hay que maximizar? ¿Por qué?
11. Seleccione un objeto de la vida cotidiana y analice sus componentes en términos de la cohesión funcional y la cohesión lógica.
12. Compare el acoplamiento entre dos unidades de programa obtenido por una instrucción `goto` con el acoplamiento obtenido mediante una llamada a procedimiento.
13. En el Capítulo 6 hemos visto que los parámetros pueden pasarse a los procedimientos por valor o por referencia. ¿Cuál de estos dos métodos proporciona la forma más compleja de acoplamiento de datos? Razone su respuesta.
14. ¿Qué problemas podrían surgir durante el mantenimiento si se desarrollara un sistema software de gran tamaño de tal manera que todos sus elementos de datos fueran globales?
15. En un programa orientado a objetos, ¿qué es lo que indica acerca del acoplamiento la declaración de una variable de instancia como pública o privada? ¿Qué razones habría para preferir declarar las variables de instancia como privadas?

***16.** Identifique un problema relacionado con el acoplamiento de datos que puede producirse en el contexto del procesamiento en paralelo.

17. Responda a las siguientes cuestiones relativas al diagrama de estructura mostrado a continuación:

a. ¿A qué módulo devuelve el control el módulo Y?

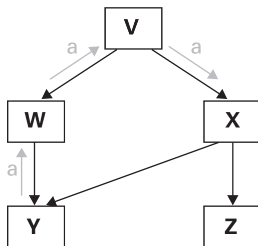
b. ¿A qué módulo devuelve el control el módulo Z?

c. ¿Están los módulos W y X enlazados mediante acoplamiento de control?

d. ¿Están los módulos W y X enlazados mediante acoplamiento de datos?

e. ¿Qué datos son compartidos por los módulos W e Y?

f. ¿De qué forma se relacionan los módulos W y Z?



18. Utilizando un diagrama de estructura, represente la estructura procedimental de un sistema simple de inventario/contabilidad para un comercio de pequeño tamaño (por ejemplo, un pequeño supermercado en una urbanización). ¿Qué módulos del sistema deberán modificarse si se efectúan cambios en las leyes que regulan los impuestos sobre las ventas? ¿Qué módulos habría que cambiar si se toma la decisión de mantener un registro de los clientes antiguos para poder enviarles por correo información comercial?

19. Utilizando un diagrama de clases, diseñe una solución orientada a objetos para el problema anterior.

20. Explique el sistema de notación de diagramas de clases de UML. Dibuje un diagrama de clases para una clase denominada `Empleado` con los miembros de datos (atributos) y métodos (operaciones) adecuados.

21. ¿Qué es el ciclo de vida de desarrollo del software? Enumere algunas características de las metodologías de desarrollo del software tradicionales.

22. Dibuje un diagrama sencillo de casos de uso que muestre las formas en las que un estudiante puede utilizar una biblioteca.

23. Dibuje un diagrama de secuencia que represente la secuencia de interacciones que tendrían lugar cuando una empresa de utilidad pública envía una factura a un cliente.

24. Dibuje un diagrama de flujo de datos simple que muestre el flujo de datos que tiene lugar en un sistema de inventario automatizado cuando se efectúa una venta.

25. ¿Por qué es necesario el modelado? ¿Cuáles son las ventajas del modelado?

26. ¿Cuál es la diferencia entre una relación uno-a-muchos y una relación muchos-a-muchos?

27. Proporcione un ejemplo de una relación uno-a-muchos que no se haya mencionado en este capítulo. Proporcione un ejemplo de una relación muchos-a-muchos que no se haya citado en este capítulo.

28. Basándose en la información de la Figura 7.10, imagine una secuencia de interacción que pueda tener lugar entre un médico y un paciente durante una visita al médico. Dibuje un diagrama de secuencia que represente esa secuencia de sucesos.

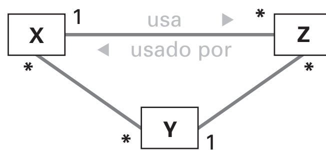
29. Dibuje un diagrama de clases que represente las relaciones entre camareros y clientes en un restaurante.

30. Dibuje un diagrama de clases que represente las relaciones entre revistas, editores de revistas y suscriptores. Incluya un conjunto de variables de instancia y los métodos de cada clase.

31. Amplíe el “diagrama de secuencia” de la Figura 7.5 para mostrar la secuencia de interacción que se produciría si el `JugadorA` devolviera correctamente la bola al

JugadorB, pero el JugadorB no consiguiera a su vez devolver la bola.

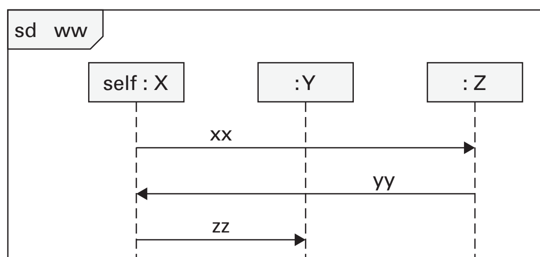
32. Responda a las siguientes preguntas basadas en el diagrama de clases que se muestra a continuación y que representa la asociación entre herramientas, sus usuarios y sus fabricantes.



a. ¿Qué clases (X, Y y Z) representan a las herramientas, los usuarios y los fabricantes? Razone su respuesta.
b. ¿Puede una herramienta ser utilizada por más de un usuario?
c. ¿Puede una herramienta ser fabricada por más de un fabricante?
d. ¿Emplea cada usuario herramientas fabricadas por muchos fabricantes?

33. En cada uno de los siguientes casos, identifique si la actividad se relaciona con un diagrama de secuencia, un diagrama de casos de uso o un diagrama de clases.
a. Representa la forma en la que los usuarios interaccionarán con el sistema.
b. Representa las relaciones entre las clases del sistema.
c. Representa la forma en la que los objetos interactuarán para llevar a cabo una tarea.

34. Responda a las siguientes cuestiones basándose en el siguiente diagrama de secuencia.



a. ¿Qué clase contiene un método denominado ww?
b. ¿Qué clase contiene un método denominado xx?
c. Durante la secuencia, ¿se comunica alguna vez directamente el objeto de “tipo” Z con el objeto de “tipo” Y?

35. Dibuje un diagrama de secuencia que indique que el objeto A llama al método bb del objeto B, que B realiza la acción solicitada y devuelve el control a A y que luego A llama al método cc del objeto B.

36. Amplíe su solución al problema anterior para indicar que A invoca al método bb solo si la variable “continuar” es verdadera y sigue llamando a bb mientras que “continuar” siga siendo verdadera después de que B devuelva el control.

37. Dibuje un diagrama de clases que muestre el hecho de que las clases `Trabajador`, `Supervisor` y `Gerente` son generalizaciones de la clase `Empleado`.

38. Basándose en la Figura 7.12, ¿qué variables de instancia adicionales estarían contenidas en un objeto de “tipo” `RegistroQuirurgico`? ¿Y en un objeto de “tipo” `RegistroVisitaMedico`?

39. Explique la relación superclase-subclase. ¿Cuál es la ventaja de este tipo de relación?

40. Explique por qué se utilizan las herramientas CASE.

41. Resuma el papel de las tarjetas CRC en la ingeniería del software.

42. ¿Hasta qué grado podríamos considerar que las estructuras de control de un lenguaje de programación de alto nivel típico (`if-then-else`, `while`, etc.) son patrones de diseño a pequeña escala?

43. ¿Cuál de las siguientes afirmaciones está relacionada con el principio de Pareto? Justifique sus respuestas.
a. Una persona maleducada puede arruinarle la fiesta a todos.
b. Cada emisora de radio se concentra en un formato concreto como por ejemplo música rock, música clásica o noticias.

c. En unas elecciones, los candidatos deberían centrar su campaña en aquel segmento del electorado que ha demostrado en un pasado su intención de ir a votar.

44. Hablando de sistemas software de gran tamaño, ¿qué es lo que esperan los ingenieros software, que esos sistemas sean homogéneos o heterogéneos en lo que al contenido de errores se refiere? Explique su respuesta.

45. ¿Cuál es la diferencia entre las pruebas alfa y las pruebas beta?

46. Describa las pruebas de caja negra.

47. ¿En qué se diferencia el desarrollo de código fuente abierto del mecanismo de pruebas beta? (Considere la diferencia entre las pruebas de caja de cristal y las pruebas de caja negra.)

48. Suponga que introducimos de manera intencionada 100 errores en un sistema software de gran tamaño antes de someter al sistema a las pruebas finales. Suponga además que se descubren y corrigen 200 errores durante estas pruebas finales, de los cuales 50 pertenecían al grupo de errores introducidos en el sistema intencionadamente. Si a continuación corregimos los siguientes 50 errores conocidos, ¿cuántos errores desconocidos estima que podría contener todavía el sistema? Explique por qué.

49. ¿Hasta qué punto es importante la documentación del usuario?

50. ¿Cuál es la ventaja de utilizar patrones de diseño?

51. Una diferencia entre la interfaz persona-computadora de un teléfono inteligente y la de una computadora de sobremesa es la relativa a la técnica utilizada para modificar la escala de una imagen en la pantalla, con el fin de obtener más o menos detalle (un proceso que se denomina “zoom”). En una computadora de sobremesa, modificamos la escala arrastrando un deslizador que está separado del área que se está visualizando o bien utilizando un menú o un elemento disponible en una barra de herramientas. En un teléfono inteligente, el cambio de escala se efectúa tocando simultáneamente la pantalla con los dedos índice y pulgar y luego modificando el espacio entre ambos puntos de contacto (un proceso que se denomina “doble toque—separar” para aumentar el grado de detalle o “doble toque—acercar” para disminuir el tamaño de la imagen).

a. Basándose en la ergonomía, ¿cómo podría justificarse esta diferencia?

b. Basándose en la cognética, ¿cómo podría justificarse esta diferencia?

52. ¿Cuál es el papel de los grupos de aseguramiento de la calidad del software dentro de las empresas dedicadas al desarrollo del software?

53. ¿De qué maneras podría fracasar un desarrollador de software a la hora de intentar obtener una patente?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. a. A un analista le asignan la tarea de implementar un sistema con el que se almacenarán registros médicos en una computadora conectada a una

red de gran tamaño. En opinión del analista, el diseño de la seguridad del sistema tiene errores, pero las preocupaciones que ha expresado son ignoradas por razones financieras. Le han dicho que continúe con el proyecto utilizando ese sistema de seguridad que cree que es inadecuado. ¿Qué debería hacer? ¿Por qué?

b. Suponga que el analista implementara el sistema tal como le han dicho y que ahora observa que esos registros médicos están siendo consultados por personal no autorizado. ¿Qué debería hacer? ¿Hasta qué punto sería ese analista responsable de ese fallo de seguridad?

c. Suponga que en lugar de obedecer a su jefe, el analista se niega a continuar con el sistema en su estado actual y da la alarma, haciendo públicos los errores que tiene el diseño, lo que provoca una serie de pérdidas financieras para la empresa y la pérdida de los empleos de muchos trabajadores inocentes. ¿Ha actuado correctamente el analista? ¿Y qué sucede si resulta que, al ser únicamente una parte del equipo global de desarrollo, ese analista no fuera consciente de que se estaban haciendo esfuerzos sinceros en algún otro lugar de la empresa para desarrollar un sistema de seguridad válido con el fin de aplicarlo al sistema en el que el analista estaba trabajando? ¿Cómo cambiaría este hecho su opinión acerca de las acciones de ese analista? (Sea consciente de que la visión que ese analista tiene de la situación es la misma que antes.)

2. Cuando un sistema software de gran tamaño está siendo desarrollado por muchas personas, ¿cómo deben distribuirse las responsabilidades legales? ¿Existe una jerarquía de responsabilidades? ¿Existen grados de responsabilidad?

3. Hemos visto que los sistemas software complejos de gran tamaño suelen ser desarrollados por muchos individuos, pocos de los cuales pueden tener una imagen completa de todo el proyecto. ¿Es éticamente correcto que un empleado contribuya al desarrollo de un proyecto sin tener un conocimiento completo de su función?

4. ¿Hasta qué grado es alguien responsable de cómo sus logros sean aplicados luego por otras personas?

5. En la relación entre un profesional de la computación y un cliente, ¿es responsabilidad del profesional implementar los deseos del cliente o influenciar esos deseos en la dirección adecuada? ¿Qué sucede si el profesional ve que los deseos de un cliente podrían tener consecuencias poco éticas? Por ejemplo, el cliente podría querer recortar en ciertas áreas en aras de la eficiencia, pero el profesional podría prever que eso sería una fuente de datos erróneos o de una mala utilización del sistema. Si el cliente insiste, ¿está libre de responsabilidad el profesional?

6. ¿Qué sucede si la tecnología comienza a avanzar tan rápidamente que los nuevos inventos quedan obsoletos antes de que el inventor haya tenido tiempo de beneficiarse de sus esfuerzos? ¿Son los beneficios materiales necesarios para motivar a los inventores? ¿Cómo relacionaría con su respuesta el éxito de los desarrollos de código fuente abierto? ¿Cree que el software libre de calidad es una realidad sostenible?

7. ¿Cree que la revolución de las computadoras está contribuyendo a resolver los problemas de energía de la humanidad? ¿Y qué sucede con otros problemas a gran escala como el hambre y la pobreza?
8. ¿Cree que los avances en la tecnología continuarán indefinidamente? ¿Cree que hay algo que podría revertir la dependencia actual que la sociedad muestra con respecto a la tecnología? ¿Cuál sería el resultado de una sociedad que continuara haciendo avanzar la tecnología indefinidamente?
9. Si tuviera una máquina del tiempo, ¿en qué periodo de la historia le gustaría vivir? ¿Hay alguna tecnología actual que le gustaría llevar consigo? ¿Puede separarse alguna tecnología de las restantes tecnologías existentes? ¿Es realista protestar contra las agresiones medioambientales y al mismo tiempo aceptar los modernos tratamientos médicos?
10. Muchas aplicaciones de un teléfono inteligente se integran automáticamente con los servicios proporcionados por otras aplicaciones. Esta integración puede hacer que una aplicación comparta con otras aplicaciones la información que se ha introducido a su través. ¿Cuáles son los beneficios de esta integración? ¿Cree que existe algún problema derivado de un "exceso" de integración?

Lecturas adicionales

Alexander, C., S. Ishikawa y M. Silverstein. *A Pattern Language*. Nueva York: Oxford University Press, 1977.

Beck, K. *Extreme Programming Explained: Embrace Change*, 2ª ed. Boston, MA: Addison-Wesley, 2004.

Bowman, D. A., E. Kruijff, J. J. LaViola, Jr. y I. Poupyrev. *3D User Interfaces Theory and Practice.* Boston, MA: Addison-Wesley, 2005.

Braude, E. *Software Design: From Programming to Architecture*. Nueva York: Wiley, 2004.

Bruegge, B. y A. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3ª ed. Boston, MA: Addison-Wesley, 2010.

Cockburn, A. *Agile Software Development: The Cooperative Game,* 2ª ed. Boston, MA: Addison-Wesley, 2006.

Fox, C. *Introduction to Software Engineering Design: Processes, Principles and Patterns with UML2.* Boston, MA: Addison-Wesley, 2007.

Gamma, E., R. Helm, R. Johnson y J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1995.

Maurer, P. M. *Component-Level Programming.* Upper Saddle River, NJ: Prentice-Hall, 2003.

Pfleeger, S. L. y J. M. Atlee. *Software Engineering: Theory and Practice*, 4ª ed. Upper Saddle River, NJ: Prentice-Hall, 2010.

Pilone, D. *UML 2.0 in a Nutshell.* Cambridge, MA: O'Reilly Media, 2005.

Pressman, R. S. *Software Engineering: A Practitioner's Approach,* 7ª ed. Nueva York: McGraw-Hill, 2009.

Schach, S. R. *Classical and Object-Oriented Software Engineering,* 8ª ed. Nueva York: McGraw-Hill, 2010.

Shalloway, A. y J. R. Trott. *Design Patterns Explained,* 2ª ed. Boston, MA: Addison-Wesley, 2005.

Shneiderman, B., C. Plaisant, Cohen, M y Jacobs S. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 5ª ed. Boston, MA: Addison-Wesley, 2009.

Sommerville, I. *Software Engineering,* 8ª ed. Boston, MA: Addison-Wesley, 2006.

CAPÍTULO

8

Abstracciones de datos

En este capítulo, vamos a investigar cómo pueden simularse otras estructuras de datos distintas de la organización en celdas proporcionada por la memoria principal de una computadora, este campo se conoce precisamente con el nombre de estructuras de datos. El objetivo es pemitir al usuario de los datos acceder a conjuntos de datos como si fueran herramientas abstractas, en lugar de obligarle a pensar en términos de la organización de la memoria principal de la computadora. Nuestro análisis mostrará cómo el deseo de construir esas herramientas abstractas conduce al concepto de objetos y de programación orientada a objetos.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

En el Capítulo 6 hemos presentado el concepto de estructura de datos, donde hemos visto que los lenguajes de programación de alto nivel proporcionan técnicas que permiten a los programadores expresar sus algoritmos como si los datos que se están manipulando estuvieran almacenados en otras formas distintas de la estructura de celdas secuenciales proporcionada por la memoria principal de una computadora. También vimos allí que las estructuras de datos soportadas por un lenguaje de programación se conocen con el nombre de estructuras primitivas. En este capítulo vamos a explorar técnicas para la construcción y manipulación de estructuras de datos distintas de las primitivas de un lenguaje, un análisis que nos llevará desde las estructuras de datos tradicionales hasta el paradigma orientado a objetos. Un tema subyacente a lo largo de toda esta progresión es la de la construcción de herramientas abstractas.

8.1 Estructuras de datos básicas

Comenzamos nuestro estudio presentando algunas estructuras básicas de datos que nos servirán como ejemplos en futuras secciones.

Arrays

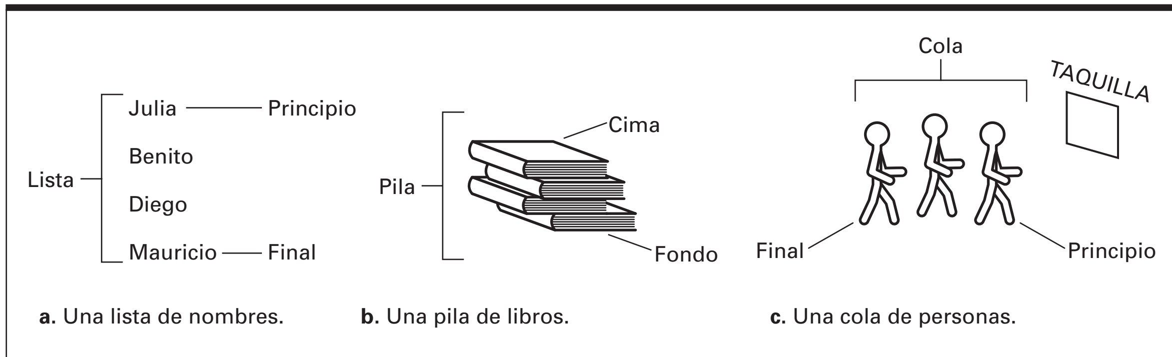
En la Sección 6.2, hemos estudiado las estructuras de datos conocidas como arrays o matrices y los tipos agregados. Recuerde que un **array** es un bloque de datos “rectangular” cuyas entradas son todas del mismo tipo. En particular, un array bidimensional está compuesto por filas y columnas en las que las distintas posiciones se identifican mediante una pareja de índices, el primer índice identifica la fila asociada con la posición, mientras que el segundo índice identifica la columna. Un ejemplo sería un array rectangular de números que representen las ventas mensuales realizadas por los distintos miembros de una empresa (las entradas de cada fila representarán las ventas mensuales realizadas por un miembro concreto del equipo, mientras que las entradas de cada columna representarán las ventas realizadas por cada miembro del equipo en un mes determinado). Así, la entrada situada en la tercera fila de la primera columna representaría las ventas realizadas por el tercer miembro del equipo en el mes de enero.

A diferencia de un array, recuerde que un **tipo agregado** o **estructura** es un conjunto de datos que pueden ser de diferentes tipos. Los elementos del conjunto se suelen denominar **componentes** o campos. Un ejemplo de estructura sería el conjunto de datos relacionado con un único empleado, cuyos componentes podrían ser el nombre del empleado (de tipo carácter), la edad (de tipo entero) y su categoría (de tipo real).

Listas, pilas y colas

Otra estructura básica de datos sería la **lista**, que es un conjunto cuyas entradas están ordenadas de manera secuencial (Figura 8.1a). El inicio de una lista se denomina **principio** de la lista. El otro extremo se denomina **final** (en ocasiones se emplean los términos cabeza y cola, pero aquí evitaremos estas denominaciones para no confundirnos con la estructura de cola).

Figura 8.1 Listas, pilas y colas.



a. Una lista de nombres. **b.** Una pila de libros. **c.** Una cola de personas.

Prácticamente cualquier conjunto de datos puede considerarse una lista. Por ejemplo, un texto puede considerarse como una lista de símbolos, una matriz bidimensional puede considerarse como una lista de filas y la música grabada en un CD puede considerarse como una lista de sonidos. Otros ejemplos más tradicionales serían las listas de invitados, las listas de la compra, las listas de matriculados en un curso y las listas de inventario. Las actividades asociadas con una lista varían dependiendo de la situación. En algunos casos, necesitaremos eliminar entradas de una lista, añadir entradas nuevas a una lista, "procesar" una a una las entradas de una lista, cambiar el orden de las entradas de la lista o quizá comprobar si un determinado elemento se encuentra dentro de una lista. Investigaremos estas operaciones posteriormente en el capítulo.

Restringiendo la forma en la que se accede a las entradas de una lista, obtenemos dos tipos de lista especiales que se conocen con los nombres de pilas y colas. Una **pila** es una lista en la que las entradas se insertan y se eliminan únicamente al principio de la lista. Un ejemplo sería una pila de libros, en la que las restricciones físicas obligan a que todas las adiciones y borrados se produzcan por la parte superior (Figura 8.1b). Siguiendo la terminología coloquial, el principio de una pila se denomina **cima** de la pila. El final de una pila se denomina **fondo** o **base**. Cuando se inserta una nueva entrada en la parte superior de una pila se dice que se **apila** (*pushing*) una entrada. Cuando se elimina una entrada de la parte superior de una pila se dice que se **desapila** (*popping*) una entrada. Observe que la última entrada colocada en una pila, será siempre la primera en ser extraída, razón por la que a las pilas se las conoce con el nombre de estructuras **LIFO** (***Last-In, First-Out;*** último en entrar, primero en salir).

Esta característica LIFO implica que una pila es ideal para almacenar elementos que deban ser extraídos en el orden inverso a aquel con el que fueron almacenados. Por ello, suelen utilizarse pilas para implementar mecanismos para deshacer actividades. Estos mecanismos se conocen con el nombre de mecanismos de **vuelta atrás** (*backtracking*) y hacen referencia al proceso de volver atrás en un sistema, en el orden opuesto con el que se entró. Un ejemplo clásico de la vida cotidiana sería el proceso de volver sobre nuestros propios pasos para salir de un bosque. Si nos vamos al campo técnico, considere por ejemplo la estructura subyacente requerida para dar soporte a un proceso recursivo. A medida que se inicia cada nueva activación, es preciso dejar momentáneamente de lado la activación anterior. Además, a medida que se

completa cada activación es preciso continuar con la última activación que se dejó de lado. Por tanto, si introducimos en una pila las activaciones a medida que las vamos dejando de lado, entonces la activación requerida se encontrará en la cima de la pila cada vez que necesitemos extraer una activación por haber completado la anterior.

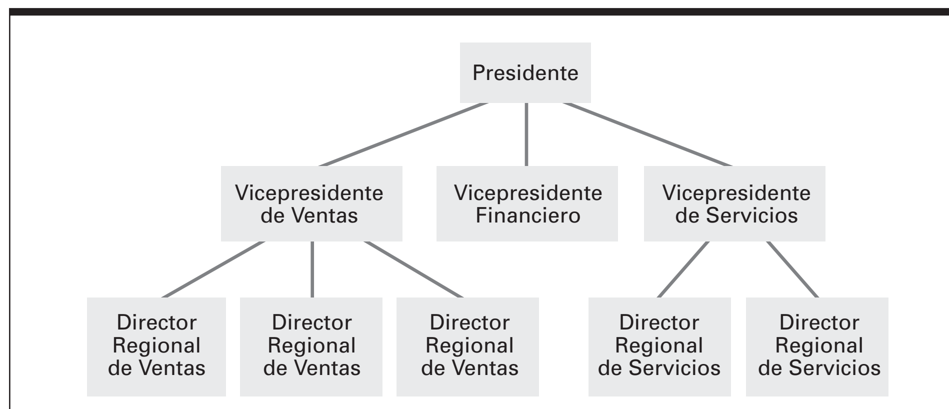
Una **cola** es una lista en la que las entradas se extraen únicamente por el principio y se insertan únicamente por el final. Un ejemplo sería una cola de personas que está esperando para comprar entradas en un cine (Figura 8.1c), el taquillero atiende a la persona que se encuentra al principio de la cola, mientras que los recién llegados se colocan al final de la misma. Ya nos hemos encontrado antes con la estructura de cola en el Capítulo 3, donde vimos que los sistemas operativos de procesamiento por lotes almacenan en una cola los trabajos en espera de ser ejecutados; esa cola se conoce como cola de trabajos. Allí vimos también que una cola es una estructura de tipo **FIFO** (***First-In, First-Out***; primero en entrar, primero en salir), lo que significa que las entradas se extraen de una cola en el mismo orden en el que fueron almacenadas.

Las colas se emplean a menudo como estructura subyacente de un buffer, que como dijimos en el Capítulo 1, es un área de memoria para el almacenamiento temporal de datos que se estén transfiriendo de una ubicación a otra. A medida que llegan los elementos de datos al buffer, se colocan al final de la cola. Después, cuando llega el momento de reenviarlos hacia su destino final, se reenvían en el orden que van apareciendo al principio de la cola. Por tanto, los elementos se reenvían en el mismo orden en el que llegaron.

Árboles

Un **árbol** es un conjunto cuyas entradas tienen una organización jerárquica similar a la del organigrama de cualquier empresa (Figura 8.2). El presidente se representa en la parte superior, con una serie de líneas ramificándose hacia abajo, hacia los vicepresidentes, que a su vez van seguidos por los directores regionales, etc. A esta definición intuitiva de la estructura de árbol vamos a imponerle una restricción adicional, que es que (en el lenguaje del organigrama de una organización) ninguna persona de la empresa tenga que informar a dos supervisores distintos. Es decir, imponemos la restricción de que las

Figura 8.2 Un ejemplo de un diagrama de una organización.



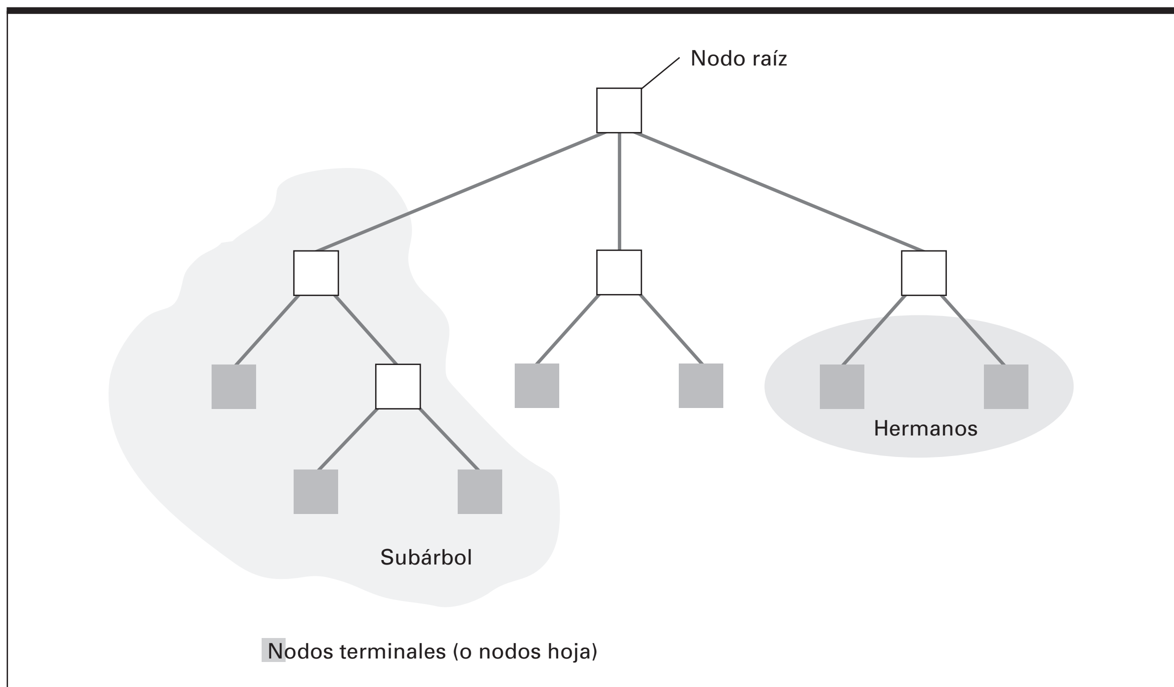
diferentes ramas de la organización no converjan en ninguno de los niveles inferiores. Ya hemos visto ejemplos de árboles en el Capítulo 6, al abordar los árboles de análisis sintáctico.

Cada posición de un árbol se denomina **nodo** (Figura 8.3). El nodo situado en la parte superior es el **nodo raíz** (si ponemos el diagrama cabeza abajo, este nodo representaría la base o raíz del árbol). Los nodos situados en el otro extremo se denominan **nodos terminales** (o, en ocasiones, **nodos hoja**). A menudo, denominaremos **profundidad** del árbol al número de nodos existente en la ruta más larga desde la raíz a una de las hojas. En otras palabras, la profundidad de un árbol es el número de niveles horizontales de que está compuesto.

En ocasiones, hablaremos de las estructuras de árbol como si cada nodo diera origen a los nodos situados inmeditamente por debajo de él. En este sentido, a menudo hablaremos de los antecesores o descendientes de un nodo. Si nos fijamos en un nodo, llamaremos **hijos** a sus descendientes inmediatos y **padre** a su antecesor inmediato. Además, diremos que dos nodos con un mismo padre son **hermanos**. Un árbol en el que cada padre no tenga más de dos hijos se denomina **árbol binario**.

Si seleccionamos cualquier nodo de un árbol, veremos que el conjunto formado por ese nodo y todos los nodos situados por debajo de él tiene también estructura de un árbol. A estas estructuras más pequeñas, pero de tipo idéntico, las llamaremos **subárbol**. Por tanto, cada nodo hijo es la raíz del subárbol existente por debajo del padre de ese nodo hijo. Cada uno de esos subárboles se dice que es una **rama** del padre. En un árbol binario, también nos referiremos a menudo a la rama izquierda o a la rama derecha de un nodo, haciendo referencia a la forma en que esté visualizándose el árbol.

Figura 8.3 Terminología de los árboles.



Cuestiones y ejercicios

1. Proporcione ejemplos (fuera de las Ciencias de la computación) de cada una de las siguientes estructuras: lista, pila, cola y árbol.
2. Indique las diferencias entre listas, pilas y colas.
3. Suponga que introducimos la letra A en una pila vacía, seguida de las letras B y C en dicho orden. Suponga que después extraemos una letra de la pila e introducimos a continuación las letras D y E. Enumere las letras que componen la pila en el orden en que aparecerían de la cima al fondo. Si extraemos una letra de la pila, ¿qué letra será?
4. Suponga que introducimos la letra A en una cola vacía, seguida de las letras B y C en dicho orden. Después, suponga que extraemos una letra de la cola e insertamos las letras D y E. Enumere las letras contenidas en la cola, en el orden en que aparecerían desde el principio al final. Si ahora extraemos una letra de la cola, ¿qué letra será?
5. Suponga que un árbol tiene cuatro nodos A, B, C y D. Si A y C son hermanos y A es el padre de D. ¿Qué nodos son nodos hoja? ¿Cuál es el nodo raíz?

8.2 Conceptos relacionados

En esta sección vamos a hablar de tres temas estrechamente relacionados con el concepto de estructuras de datos: la abstracción, la diferencia entre estructuras estáticas y dinámicas, y el concepto de puntero.

Otra vez la abstracción

Las estructuras presentadas en la sección anterior suelen estar asociadas con datos. Sin embargo, la memoria principal de una computadora no está organizada como arrays, listas, pilas, colas y árboles, sino como una secuencia de celdas de memoria direccionables. Por tanto, todos los demás tipos de estruc-turas deberán ser simulados. El tema de este capítulo es precisamente cómo llevar a cabo esa simulación. Por ahora, apuntemos simplemente que las organizaciones de datos como matrices, listas, pilas, colas y árboles son organizaciones abstractas de los datos que se crean para ocultar a ojos de los usuarios de los datos los detalles del almacenamiento real de esos datos y para que esos usuarios puedan acceder a la información como si esta estuviera almacenada de una forma más conveniente.

En este contexto, el término ***usuario*** no hace referencia necesariamente a una persona. En lugar de ello, el significado de la palabra dependerá de cuál sea nuestra perspectiva en cada momento. Si pensamos en términos de una persona que usa un PC para mantener sus registros de la liga de fútbol del colegio, entonces el usuario será una persona. En este caso, el software de aplicación (quizá un paquete de hoja de cálculo) será responsable de presentar los datos en una forma abstracta que sea cómoda para esa persona, lo más probable es que los presente en forma de matriz. Si pensamos en términos de un servidor

conectado a Internet, entonces el usuario podría ser una máquina cliente. En este caso, el servidor sería responsable de presentar los datos en una forma abstracta que sea cómoda para el cliente. Si estamos pensando en términos de la estructura modular de un programa, entonces el usuario sería cualquier módulo que necesite acceder a los datos. En este caso, el módulo que contenga los datos será responsable de presentarlos de una forma abstracta que sea cómoda para los demás módulos. En cada uno de estos escenarios, lo importante es que el usuario tiene el privilegio de acceder a los datos mediante una representación abstracta.

Estructuras de datos estáticas y dinámicas

Una diferencia importante a la hora de construir estructuras de datos abstractas es si la estructura que se está simulando es estática o dinámica; es decir, si la forma o el tamaño de la estructura varía a lo largo del tiempo. Por ejemplo, si la abstracción es una lista de nombres, es importante considerar si la lista tendrá un tamaño fijo a lo largo de toda su existencia o se va a ir ampliando y contrayendo a medida que se añadan o se borren nombres.

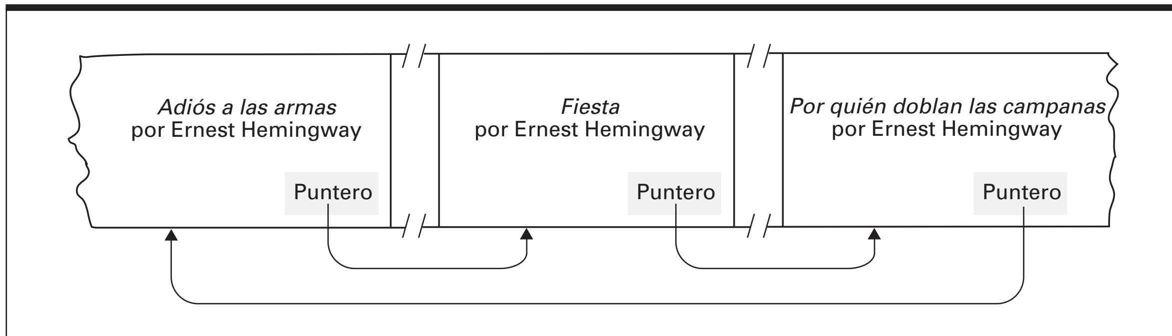
Como regla general, las estructuras de datos estáticas se manipulan más fácilmente que las dinámicas. Si una estructura es estática, simplemente necesitamos proporcionar un medio para acceder a los distintos elementos de datos de la estructura, y quizá también un medio de modificar los valores almacenados en ciertas ubicaciones. Pero si la estructura es dinámica, tendremos que ocuparnos también de los problemas de añadir y eliminar entradas, así como con el de localizar el espacio de memoria requerido por una estructura de datos cuyo tamaño crezca. Si tenemos una estructura mal diseñada, el añadir una única entrada nueva podría provocar que numerosos elementos de la estructura tuvieran que ser recolocados, y un crecimiento excesivo de la misma podría obligar a que toda la estructura se transfiriera a otra área de memoria en la que hubiera disponible más espacio.

Punteros

Recuerde que las distintas celdas de la memoria principal de una máquina se identifican mediante direcciones numéricas. Al ser valores numéricos, estas direcciones pueden también codificarse y almacenarse en celdas de memoria. Un **puntero** es un área de almacenamiento que contiene una de esas direcciones codificadas. En el caso de las estructuras de datos, se utilizan punteros para anotar la ubicación en la que están almacenados los campos de datos. Por ejemplo, si debemos mover repetidamente un elemento de datos de una ubicación a otra, podríamos elegir una ubicación fija para que actuara como puntero. Entonces, cada vez que movamos el elemento, podemos actualizar el puntero con el fin de reflejar la nueva dirección en la que ese dato se encuentra. Posteriormente, cuando necesitemos acceder a ese elemento de datos, podremos localizarlo por medio del puntero. De hecho, el puntero se llama así porque siempre “apunta” al dato.

Ya nos hemos encontrado con el cocepto de puntero al analizar los procesadores en el Capítulo 2. Allí vimos que se utiliza un registro denominado contador de programa para almacenar la dirección de la siguiente instrucción que

Figura 8.4 Novelas ordenadas por título, pero enlazadas según el autor.



hay que ejecutar. Por tanto, el contador de programa desempeña el papel de un puntero. De hecho, al contador de programa se le denomina también en ocasiones **puntero de instrucciones**.

Como ejemplo de la aplicación de punteros, suponga que tenemos almacenada en la memoria de una computadora una lista de novelas ordenadas alfabéticamente según el título. Aunque es adecuada para muchas aplicaciones, esta ordenación hace difícil encontrar todas las novelas de un determinado autor, ya que están dispersas por toda la lista. Para resolver este problema, podemos reservar una celda de memoria adicional dentro de cada uno de los bloques de celdas que representen a una novela y usar esa celda adicional como puntero a otro bloque que también se corresponda con un libro del mismo autor. De esta forma, las novelas que tengan un mismo autor pueden enlazarse en bucle (Figura 8.4). Una vez que hemos encontrado una novela escrita por un autor determinado, podemos encontrar todas las restantes siguiendo los punteros de un libro a otro.

Muchos lenguajes de programación modernos incluyen los punteros como tipo de datos primitivo. Es decir, permiten la declaración, asignación y manipulación de punteros en formas similares a como se haría con datos de tipo entero o con cadenas de caracteres. Utilizando un lenguaje de este tipo, un programador puede diseñar redes de datos complejas dentro de la memoria de una máquina, en la que se utilizan punteros para enlazar entre sí los elementos de datos relacionados.

Cuestiones y ejercicios

1. ¿En qué sentido son abstracciones las estructuras de datos tales como matrices, listas, pilas, colas y árboles?
2. Describa una aplicación en la que cabría esperar que se utilizara una estructura de datos estática. A continuación describa una aplicación en la que cabría esperar que se empleara una estructura de datos dinámica.
3. Describa algunos contextos, ajenos a las Ciencias de la computación, en los que aparezca el concepto de puntero.

8.3 Implementación de estructuras de datos

Analicemos ahora las maneras en las que pueden almacenarse en la memoria principal de una computadora las estructuras de datos de las que hemos hablado en la sección anterior.

Almacenamiento de arrays

Comencemos con las técnicas para almacenar matrices. Como hemos visto en el Capítulo 6, estas estructuras suelen proporcionarse como estructuras primitivas en los lenguajes de programación de alto anivel. Nuestro objetivo aquí es comprender cómo se traducen los programas que manejan esas estructuras a programas en lenguaje máquina que manipulan los datos almacenados en la memoria principal.

Matrices Suponga que queremos almacenar una secuencia de 24 mediciones de temperatura, cada una de las cuales requiere una celda de memoria de espacio de almacenamiento. Suponga además que queremos idenficar esas medidas según su posición dentro de la secuencia. Es decir, queremos poder acceder, por ejemplo, a la primera o a la quinta de esas medidas. En resumen, queremos poder manipular la secuencia como si fuera una matriz homogénea unidimensional.

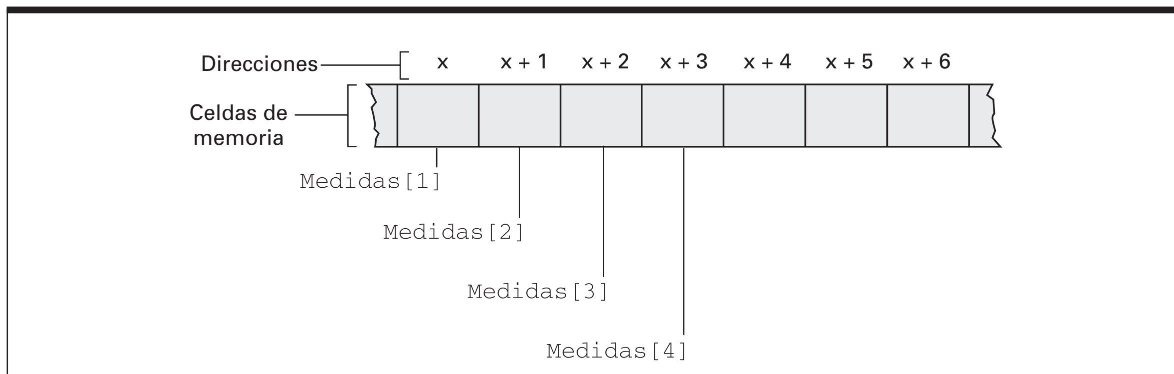
Podemos conseguir este objetivo simplemente almacenando las medidas en una secuencia de 24 celdas de memoria con direcciones consecutivas. Entonces, si la dirección de la primera celda de la secuencia es x, la posición de cualquier medida de temperatura concreta puede calcularse restando una unidad del índice de la medida deseada y luego sumando el resultado a x. En particular, la cuarta medida estará ubicada en la dirección $x + (4 - 1)$, como se muestra en la Figura 8.5.

La mayoría de los traductores de los lenguajes de programación de alto nivel utilizan esta técnica para implementar matrices unidimensionales. Cuando el traductor se encuentra una instrucción de declaración tal como

```
int Medidas[24];
```

que declara que el término `Medidas` hace referencia a una matriz unidimensional de 24 valores enteros, el traductor se las arregla para reservar 24 celdas con-

Figura 8.5 La matriz de medidas de temperatura almacenada en memoria, comenzando en la dirección x.



secutivas. Posteriormente en el programa, si se encuentra con la instrucción de asignación

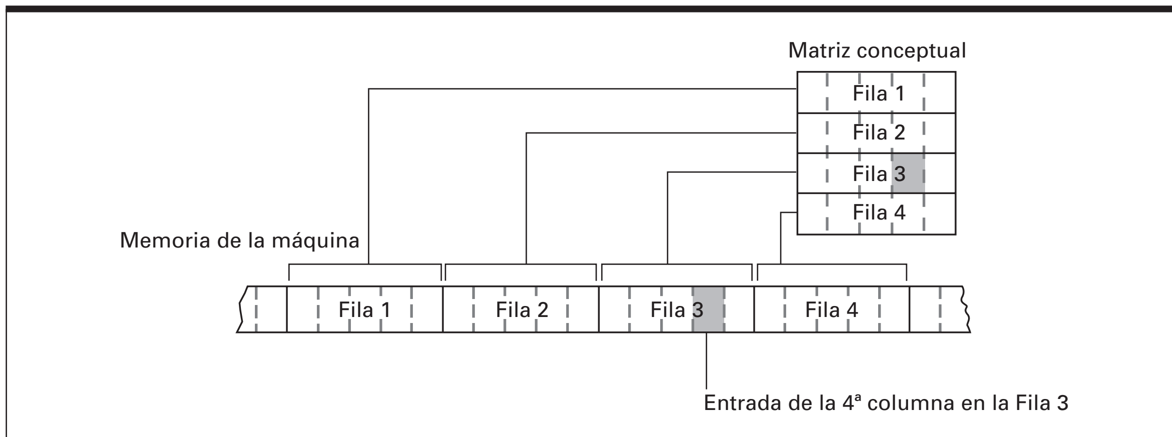
```
Medidas[4] ← 67;
```

que solicita que se almacene el valor 67 en la cuarta entrada de la matriz `Medidas`, el traductor construye la secuencia de instrucciones máquina requeridas para colocar el valor 67 en la celda de memoria situada en la posición $x + (4 - 1)$, donde x es la dirección de la primera celda del bloque asociado con la matriz `Medidas`. De esta manera, el programador puede escribir el programa como si las medidas de temperatura estuvieran realmente almacenadas en una matriz unidimensional. (Precuación: en los lenguajes C, C++, C# y Java, los índices de las matrices comienzan en 0 en lugar de en 1, por lo que para referirnos a la cuarta medida tendríamos que utilizar `Medidas[3]`. Véase la Cuestión/ejercicio 3 al final de esta sección.)

Suponga ahora que queremos almacenar las ventas realizadas por el equipo comercial de una empresa a lo largo de un periodo de una semana. En este caso, podríamos estructurar los datos en una matriz homogénea bidimensional, en la que los valores de cada fila especificaran las ventas realizadas por un empleado concreto y los valores de cada columna representaran todas las ventas realizadas en un día en particular.

Para conseguir esto, lo primero que hacemos es darnos cuenta de que la matriz es estática, en el sentido de que su tamaño no varía a medida que se realizan actualizaciones. Podemos por tanto calcular la cantidad de área de almacenamiento que necesitamos para toda la matriz y reservar un bloque contiguo de celdas de memoria del tamaño requerido. A continuación, almacenamos los datos en la matriz fila a fila. Comenzando con la primera celda del bloque que hemos reservado, almacenamos los valores de la primera fila de la matriz en posiciones consecutivas de memoria, después almacenamos la siguiente fila, luego la siguiente, y así sucesivamente (Figura 8.6). De un almacenamiento de este estilo diríamos que utiliza **ordenación por filas**, a diferencia de la **ordenación por columnas** en la que la matriz se almacenaría columna a columna.

Figura 8.6 Una matriz bidimensional con cuatro filas y cinco columnas almacenada utilizando ordenación por filas.



Con los datos almacenados de esta manera, veamos cómo podríamos localizar el valor correspondiente a la tercera fila y la cuarta columna de la matriz. Imagine que nos encontramos en la primera posición del bloque de memoria de la máquina que hemos reservado. Comenzando por esa posición, lo primero que nos encontramos son los datos de la primera fila de la matriz, seguidos de los correspondientes a la segunda fila, luego a la tercera, y así sucesivamente. Para llegar a la tercera fila, debemos pasar antes por la primera y la segunda filas. Puesto que cada fila contiene cinco entradas (una por cada día laborable de la semana, de lunes a viernes), tenemos que dejar atrás un total de 10 entradas para poder llegar a la primera entrada de la tercera fila. A partir de ahí, deberemos dejar atrás otras tres entradas para alcanzar la entrada correspondiente a la cuarta columna de esa fila. En total, para llegar a la entrada de la tercera fila y la cuarta columna, debemos saltarnos 13 entradas, contando desde el principio del bloque.

El cálculo anterior puede generalizarse para obtener una fórmula que permita convertir las referencias expresadas en términos de posiciones de fila y columna a direcciones reales de memoria. En particular, si representamos mediante c el número de columnas de una matriz (que equivale al número de entradas de cada fila), entonces la dirección de la entrada situada en la i-ésima fila y la j-ésima columna será

$$x + (c \times (i - 1)) + (j - 1)$$

donde x es la dirección de la celda que contiene la entrada correspondiente a la primera fila y la primera columna de la matriz. Es decir, debemos saltar $i - 1$ filas, cada una de las cuales contiene c entradas, hasta alcanzar la i-ésima fila y luego deberemos saltar $j - 1$ entradas más con el fin de alcanzar la j-ésima de esa fila. En nuestro ejemplo anterior, $c = 5$, $i = 3$ y $j = 4$, por lo que si la matriz estuviera almacenada comenzando en la dirección x, entonces la entrada de la tercera fila y la cuarta columna estaría en la dirección $x + (5 \times (3 - 1)) + (4 - 1) = x + 13$. La expresión $(c \times (i - 1)) + (j - 1)$ en ocasiones se denomina **polinomio de dirección** (*address polynomial*).

Una vez más, esta es la técnica utilizada por la mayor parte de los traductores de los lenguajes de programación de alto nivel. Al encontrarse con la instrucción de declaración

```
int Ventas[8, 5];
```

que declara que el término `Ventas` hace referencia a una matriz bidimensional de valores enteros con 8 filas y 5 columnas, el traductor reserva 40 celdas de memoria consecutivas. Posteriormente, si se encuentra con la instrucción de asignación

```
Ventas[3, 4] ← 5;
```

que solicita que se almacene el valor 5 en la entrada correspondiente de la tercera fila y la cuarta columna de la matriz `Ventas`, construye la secuencia de instrucciones máquina requeridas para colocar el valor 5 en la celda de memoria cuya dirección es $x + 5 \times (3 - 1) + (4 - 1)$, donde x es la dirección de la primera celda del bloque de memoria asociado con la matriz `Ventas`. De esta forma, el programador puede escribir el programa como si los datos de ventas estuvieran realmente almacenados en una matriz bidimensional.

Implementación de listas contiguas

Las primitivas para la construcción y manipulación de matrices que se proporciona en la mayoría de los lenguajes de programación de alto nivel son herramientas cómodas para la construcción y manipulación de listas contiguas. Si las entradas de la lista son todas ellas del mismo tipo de datos primitivo, entonces no es nada más que una matriz homogénea unidimensional. Un ejemplo algo más complejo sería una lista de diez nombres, cada uno de ellos con no más de ocho caracteres, como se analiza en el texto. En este caso, un programador podría construir la lista contigua en forma de una matriz bidimensional de caracteres con diez filas y ocho columnas, que nos daría la estructura representada en la Figura 8.6 (suponiendo que la matriz se almacenara con ordenación por filas).

Muchos lenguajes de alto nivel incorporan características que animan a utilizar este tipo de implementación para las listas. Por ejemplo, suponga que llamáramos `ListaMiembros` a la matriz bidimensional de caracteres que acabamos de proponer. Entonces, además de la notación tradicional en la que la expresión tradicional `ListaMiembros [3,5]` hace referencia al carácter individual situado en la tercera fila y la quinta columna, algunos lenguajes adoptan la expresión `ListaMiembros [3]` para hacer referencia a la tercera fila completa, lo que se correspondería con la tercera entrada de la lista.

Estructuras Suponga ahora que deseamos almacenar una estructura denominada `Empleado`, formada por los tres campos siguientes: `Nombre` de tipo carácter, `Edad` de tipo entero y `Categoria` de tipo real. Si el número de celdas de memoria requerido por cada campo es fijo, entonces podemos almacenar la matriz en un bloque de celdas contiguas. Por ejemplo, suponga que el campo `Nombre` requiere como máximo 25 celdas, `Edad` requiere una única celda y `Categoria` precisa solo una celda. Entonces, podríamos reservar un bloque de 27 celdas contiguas para almacenar el nombre en las 25 primeras, la edad en la celda número 26 y la categoría en la última celda (Figura 8.7a).

Con esta ordenación, sería fácil acceder a los distintos campos dentro de la estructura. Si la dirección de la primera celda fuera x, entonces cualquier referencia a `Empleado.Nombre` (es decir, al campo `Nombre` de la estructura `Empleado`) se traduciría a las 25 celdas que comienzan en la dirección x, mientras que una referencia a `Empleado.Edad` (el componente `Edad` de `Empleado`) se traduciría a la celda situada en la dirección $x + 25$. En particular, si un traductor se encontrara con una sentencia como

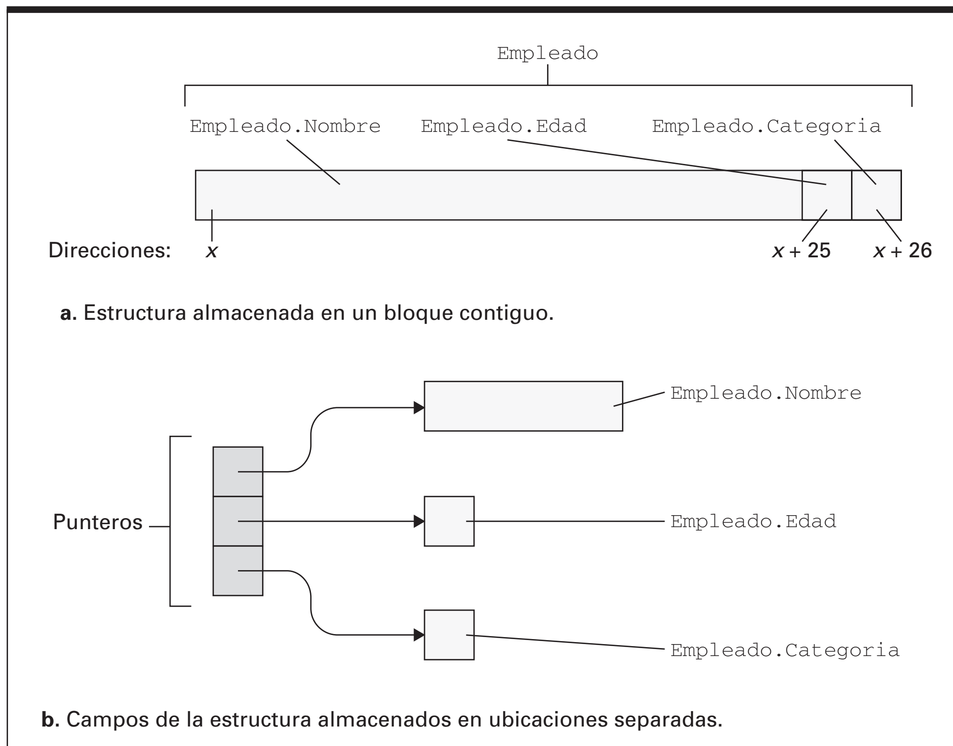
```
Empleado.Edad ← 22;
```

en un programa de alto nivel, entonces simplemente construiría las instrucciones de lenguaje máquina necesarias para almacenar el valor 22 en la celda de memoria cuya dirección fuera $x + 25$. O bien, si definimos `EmpleadoDelMes` como una estructura similar almacenada en la dirección y, entonces la sentencia

```
EmpleadoDelMes ← Empleado;
```

se traduciría a una secuencia de instrucciones que copiaría el contenido de las 27 celdas que comienzan en la dirección x a las 27 celdas que comienzan en la dirección y.

Figura 8.7 Almacenamiento de la estructura **`Empleado`**.



Una alternativa para almacenar una estructura en un bloque de celdas de memoria contiguas consiste en almacenar cada componente en una ubicación distinta y luego enlazarlas por medio de punteros. Para ser más precisos, si la matriz contiene tres componentes, entonces reservamos primero un lugar de la memoria para almacenar tres punteros, cada uno de los cuales apuntará a uno de los campos (Figura 8.7b). Si estos punteros se almacenan en un bloque que comienza en la dirección x, entonces podremos encontrar el primer componente siguiendo el puntero almacenado en la posición x, el segundo componente siguiendo el puntero situado en la posición $x + 1$, etc.

Esta disposición de los datos es especialmente útil en aquellos casos en los que el tamaño de los componentes de una estructura es dinámico. Por ejemplo, utilizando el sistema de punteros, podemos incrementar el tamaño del primer componente simplemente encontrando un área de la memoria en la que almacenar el nuevo valor del componente (que ahora es más largo) y luego ajustando el puntero apropiado, para que apunte a esa nueva ubicación. Sin embargo, si la estructura estuviera almacenada en un bloque de memoria contiguo, sería necesario modificar la estructura completa.

Almacenamiento de listas

Consideremos ahora las técnicas para almacenar una lista de nombres en la memoria principal de una computadora. Una de las estrategias posibles sería almacenar la lista completa en un mismo bloque de celdas de memoria con

direcciones consecutivas. Suponiendo que cada nombre no tenga más de ocho caracteres, podemos dividir ese bloque de celdas en un conjunto de subbloques, teniendo cada uno de ellos ocho celdas. En cada sub-bloque podemos almacenar un nombre escribiendo su código ASCII usando una celda por cada letra. Si el nombre es muy corto y no ocupa todas las celdas del subbloque que tiene asignado, simplemente podemos rellenar las restantes celdas con el código ASCII correspondiente al carácter de espacio. El uso de este sistema requiere un bloque de 80 celdas de memoria consecutivas para almacenar una lista de 10 nombres.

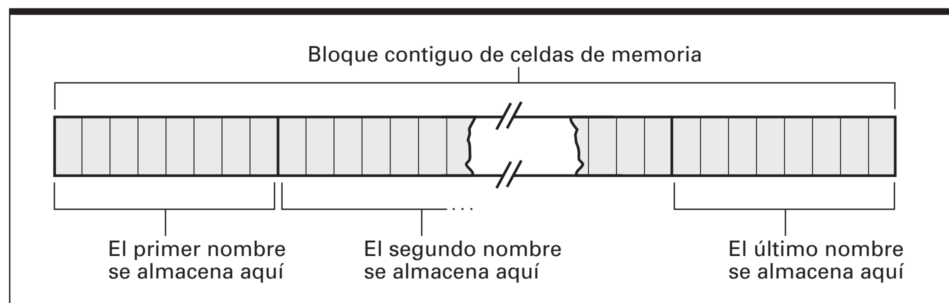
El sistema de almacenamiento que acabamos de describir se ilustra en la Figura 8.8. Lo más importante es que toda la lista se almacena en un mismo bloque de memoria de gran tamaño, estando las entradas sucesivas unas a continuación de otras en celdas de memoria contiguas. Este tipo de organización se denomina **lista contigua**.

Una lista contigua es una estructura de almacenamiento bastante conveniente para implementar listas estáticas, pero tiene sus desventajas en el caso de listas dinámicas, en las que el borrado y la inserción de nombres puede hacer que haya que dedicar un tiempo considerable a mover las entradas de sitio. En un escenario de caso peor, la adición de entradas obligaría a mover toda la lista a una nueva ubicación con el fin de disponer de un bloque de celdas consecutivas lo suficientemente grande como para albergar la lista ampliada.

Estos problemas pueden simplificarse si permitimos que las entradas individuales de una lista se almacenen en áreas de memoria diferentes, en lugar de almacenarlas todas juntas en un mismo bloque de gran tamaño. Para tratar de explicarnos, pensemos de nuevo en el ejemplo de almacenamiento de una lista de nombres (en el que cada nombre no tiene más de ocho caracteres de longitud). Esta vez vamos a almacenar cada nombre en un bloque de nueve celdas contiguas. Las primeras ocho de estas celdas se utilizan para almacenar el propio nombre, mientras que la última celda se emplea como puntero al siguiente nombre de la lista. Con esta técnica, la lista puede estar dispersa por la memoria principal, ocupando varios pequeños bloques de nueve celdas que estarán enlazados entre sí mediante punteros. A causa de este sistema de enlazamiento, a dicho tipo de organización se le denomina **lista enlazada**.

Para saber dónde comienza una lista enlazada, definimos otro puntero en el que almacenamos la dirección de la primera entrada. Puesto que este puntero apunta al principio de la lista se le denomina **puntero de cabecera**.

Figura 8.8 Nombres almacenados en la memoria en forma de lista contigua.



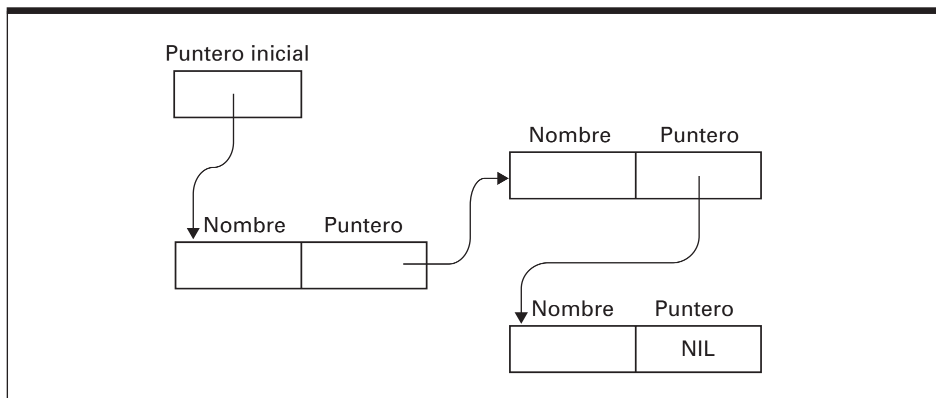
Para marcar el final de una lista enlazada, utilizamos lo que se denomina un **puntero NIL** (también denominado **puntero nulo**), que es simplemente un patrón de bits especial que se almacena en la celda correspondiente al puntero de la última entrada para indicar que ya no hay más entradas en la lista. Por ejemplo, si acordamos desde el principio que nunca vamos a almacenar una entrada de una lista en la dirección 0, el valor cero nunca aparecerá como valor de un puntero legítimo, así que podemos emplearlo como puntero NIL.

Esta estructura final de lista enlazada se muestra en el diagrama de la Figura 8.9, en el que se representan mediante rectángulos individuales los bloques de memoria dispersos utilizados por la lista. Cada uno de los rectángulos está etiquetado para indicar su composición. Cada puntero se representa mediante una flecha que va desde el propio puntero hasta la celda a la que apunta. Para recorrer la lista, primero tenemos que seguir el puntero de cabecera o inicial, con el fin de localizar la primera entrada. A partir de ahí, vamos siguiendo los punteros almacenados junto con las entrdas, saltando de una entrada a la siguiente hasta llegar al puntero NIL.

Para ver las ventajas de una lista enlazada con respecto a una lista contigua, vamos a ver qué pasaría en el caso de que hubiera que borrar una entrada. En el caso de una lista contigua, esto daría lugar a la aparición de un hueco, lo que quiere decir que todas las entradas situadas a continuación de la que acabamos de borrar tendrían que moverse hacia adelante, para que la lista continuara siendo contigua. Sin embargo, en el caso de una lista enlazada, podemos borrar una entrada simplemente cambiando un único puntero. Esto se hace modificando el puntero que apuntaba originalmente a la entrada borrada, de modo que ahora pase a apuntar a la entrada situada a continuación de la que hemos borrado (Figura 8.10). A partir de ese momento, cuando se recorra la lista, nos saltaremos la entrada borrada porque ya no forma parte de la cadena.

La inserción de una nueva entrada en una lista enlazada es solo un poquito más complicada. Primero tenemos que utilizar un bloque no utilizado de celdas de memoria que sea lo suficientemente grande como para poder almacenar la nueva entrada, junto con un puntero. En ese bloque almacenaremos la nueva entrada y rellenaremos el puntero con la dirección de la entrada de la lista que debe estar situada a continuación de la nueva entrada. Por último, modificaremos el puntero asociado con la entrada que debe estar situada inmediatamente

Figura 8.9 Estructura de una lista enlazada.



Un problema con los punteros

Al igual que el uso de diagramas de flujo conduce a diseños de algoritmos muy complicados (Capítulo 5) y el uso indiscriminado de sentencias `goto` hace que se obtengan programas mal diseñados (Capítulo 6), el uso inadecuado de punteros puede producir estructuras de datos innecesariamente complejas y proclives a errores. Para poner orden en este caos, muchos lenguajes de programación restringen la flexibilidad de los punteros. Por ejemplo, Java no permite utilizar punteros en su forma general. En lugar de ello, solo permite una forma restringida de punteros denominados referencias. Una distinción entre referencias y punteros es que las referencias no pueden modificarse mediante operaciones aritméticas. Por ejemplo, si un programador de Java quiere hacer avanzar la referencia `Siguiente` hasta la siguiente entrada de una lista contigua, utilizaría una sentencia equivalente a

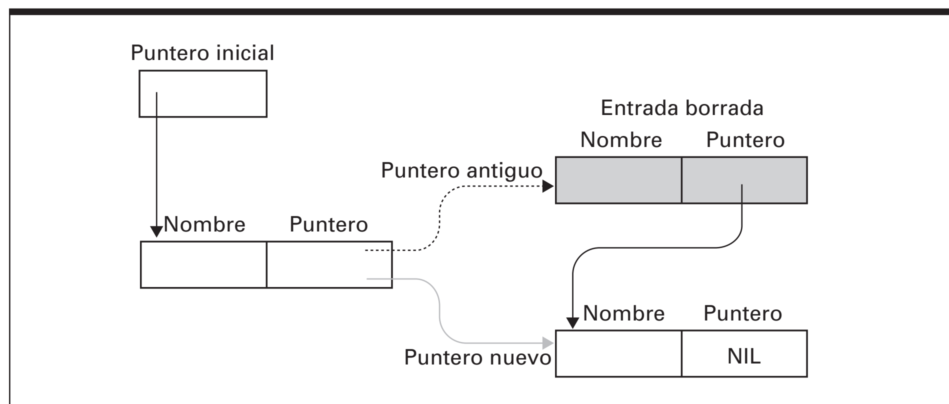
```
redirigir Siguiente a la siguiente entrada de la lista
```

mientras que un programador de C utilizaría una sentencia equivalente a

```
asignar a Siguiente el valor Siguiente + 1
```

Observe que la sentencia de Java refleja mejor cuál es el objetivo subyacente. Además, para ejecutar la sentencia Java, debe existir otra entrada en la lista, mientras que si `Siguiente` ya estuviera apuntando a la última entrada de la lista, la instrucción de C haría que `Siguiente` apuntara a algo que está situado fuera de la lista, un error que los programadores de C principiantes, e incluso los experimentados, cometen de manera bastante frecuente.

Figura 8.10 Borrado de una entrada en una lista enlazada.

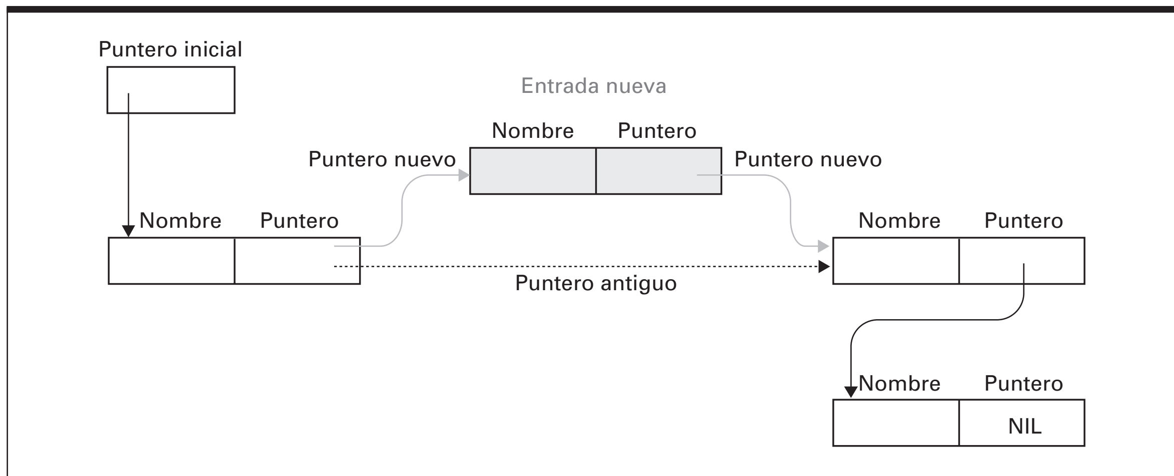


antes de la nueva entrada, de modo que ahora apunta a la nueva entrada (Figura 8.11). Después de llevar a cabo esta modificación, la nueva entrada se encontrará en el lugar apropiado cada vez que recorramos la lista.

Almacenamiento de pilas y colas

Para almacenar pilas y colas se suele emplear una organización similar a la de las listas contiguas. En el caso de una pila, se reserva un bloque de memoria lo suficientemente grande como para que quepa la pila con su tamaño máximo (la

Figura 8.11 Inserción de una entrada en una lista enlazada.



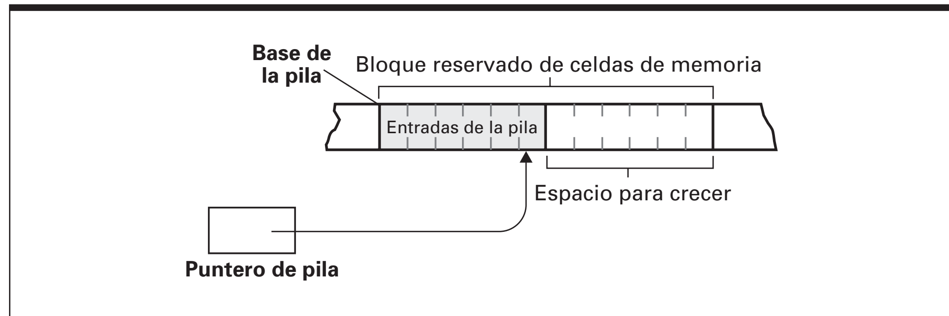
determinación del tamaño de este bloque puede ser, en muchas ocasiones, una decisión de diseño crítica. Si se reserva demasiado poco espacio, la pila terminará por sobrepasar el espacio de almacenamiento asignado; sin embargo, si se reserva demasiado espacio se estarán desperdiciando celdas de memoria.) Uno de los extremos de este bloque se designa como base de la pila. Es ahí donde se almacenará la primera entrada que se introduzca en la pila. Después, cada entrada adicional se colocará a continuación de su predecesora, a medida que la pila vaya creciendo hacia el otro extremo del bloque de memoria reservado.

Observe que a medida que se introducen y se extraen entradas en la pila, la ubicación de la cima de la pila se irá moviendo hacia atrás y hacia adelante dentro del bloque de celdas de memoria reservado. Para saber en todo momento dónde está ubicada la cima de la pila, almacenamos su dirección en una celda de memoria adicional, que se conoce con el nombre de **puntero de pila**; el puntero de pila es un puntero que indica dónde está situada la cima de la pila.

El sistema completo, ilustrado en la Figura 8.12, funciona de la manera siguiente: para añadir una nueva entrada a la pila, primero ajustamos el puntero de pila de manera que ahora apunte al espacio libre situado justo detrás de la cima de la pila y luego colocamos dicha entrada en esa ubicación. Para extraer una entrada de la pila, leemos los datos a los que apunta el puntero de pila y luego ajustamos el valor del puntero de pila, de forma que ahora apunte a la entrada de la pila situada justo debajo.

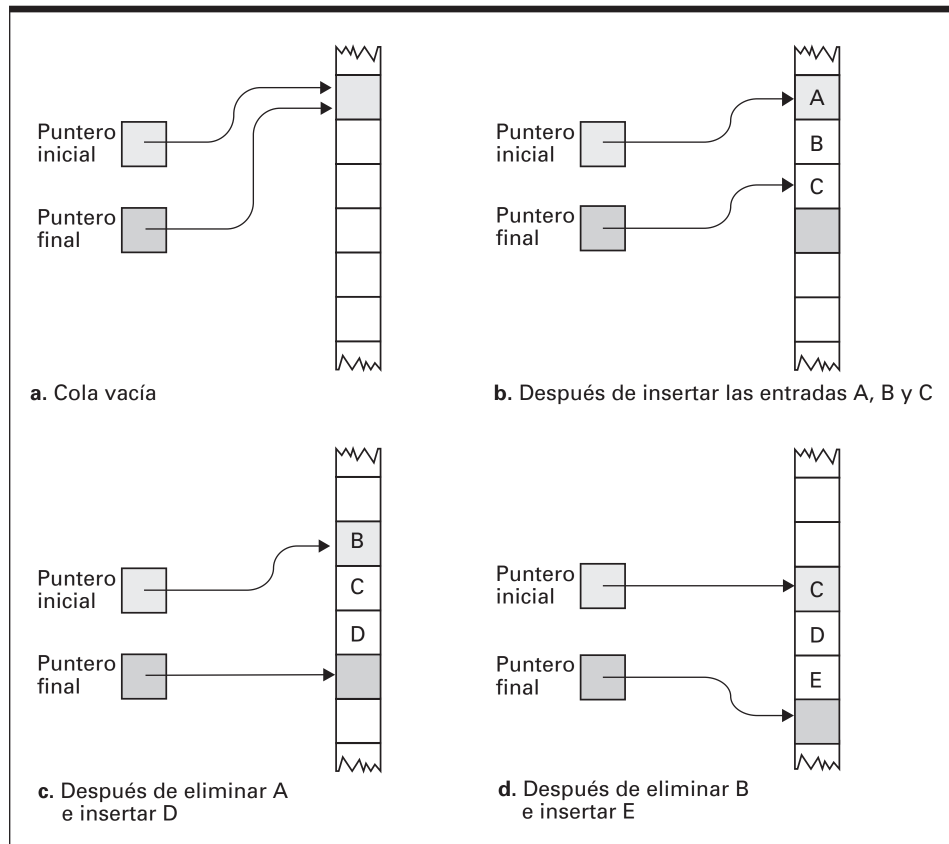
La implementacion tradicional de una cola es similar a la de una pila. De nuevo, reservamos un bloque de celdas de memoria contiguas en la memoria principal, lo suficientemente grande como para que quepa la cola cuando alcance el tamaño máximo que hayamos previsto. Sin embargo, en el caso de una cola, tenemos que realizar operaciones en ambos extremos de la estructura, por lo que reservamos dos celdas de memoria que utilizaremos como punteros, en lugar de reservar solo una, como hacíamos con las pilas. Uno de esos punteros, denominado **puntero de cabecera** (o puntero inicial), indica dónde se encuentra el principio de la cola; el otro, denominado **puntero final**, indica dónde termina la cola. Cuando la cola está vacía, ambos punteros apuntan a la

Figura 8.12 Una pila en la memoria.



misma ubicación (Figura 8.13). Cada vez que se inserta una entrada en la cola, se coloca en la ubicación a la que apunta el puntero final, y luego este puntero se ajusta para que apunte a la siguiente posición no utilizada. De esta forma, el puntero final está siempre apuntando a la primera celda libre del final de la cola. Para extraer una entrada de la cola, tenemos que extraer la entrada a la que apunta el puntero de cabecera y luego ajustar dicho puntero para que apunte a la siguiente entrada de la cola.

Figura 8.13 Implementación de una cola mediante sendos punteros inicial y final. Observe que la cola se desplaza por la memoria a medida que se insertan y se eliminan entradas.

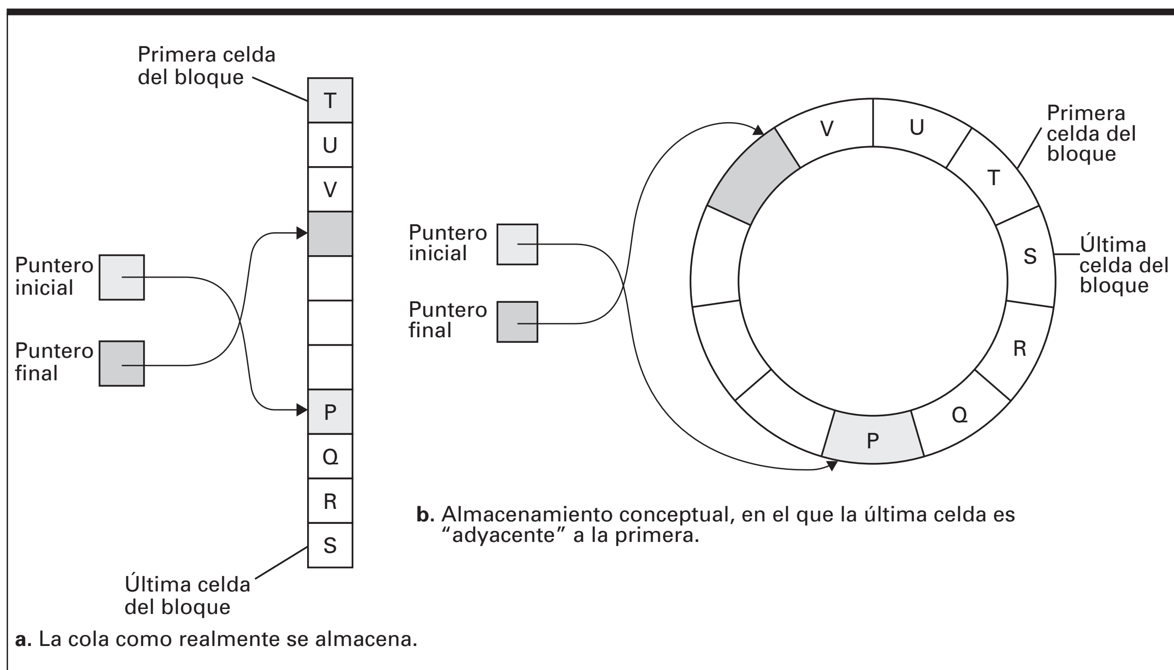


Un problema con el almacenamiento de colas que hemos descrito hasta ahora es que a medida que se van insertando y eliminando entradas en la cola, esta se va desplazando por la memoria como si fuera un glaciar (véase de nuevo la Figura 8.13). Por tanto, necesitamos un mecanismo para confinar la cola dentro de los límites de su bloque de memoria reservado. La solución es muy simple. Lo que hacemos es permitir que la cola vaya migrando a lo largo del bloque. Entonces, cuando el final de la cola llegue al final del bloque, comenzamos a insertar las entradas adicionales por el principio del bloque, que a estas alturas estará vacío. De la misma manera, cuando la última entrada del bloque llegue a convertirse en el principio de la cola y esta entrada sea extraída, ajustaremos el puntero inicial para que vuelva a apuntar al principio del bloque, donde ya habrá otras entradas esperando. De esta forma, la cola va "persiguiéndose" a sí misma por el bloque, como si los extremos del bloque estuvieran conectados formando un bucle (Figura 8.14). El resultado es una implementación que se conoce con el nombre de **cola circular**.

Almacenamiento de árboles binarios

Para analizar las técnicas de almacenamiento de árboles, vamos a fijarnos tan solo en los árboles binarios que, como recordará, son árboles en los que cada nodo tiene como máximo dos hijos. Dichos árboles suelen almacenarse en memoria utilizando una estructura enlazada similar a la que se emplea con las listas enlazadas. Sin embargo, en lugar de estar cada entrada formada por dos componentes (los datos seguidos de un puntero a la siguiente entrada), cada entrada (o nodo) del árbol binario está formada por tres componentes: (1) los datos, (2) un puntero al primer hijo de ese nodo y (3) un puntero al segundo

Figura 8.14 Una cola circular que contiene las letras P hasta V.



hijo de ese nodo. Aunque no existen los conceptos de izquierda y derecha en una máquina, en ocasiones es útil referirse al primer puntero como el **puntero al hijo izquierdo** y al otro puntero como **puntero al hijo derecho**, haciendo referencia a la forma en que dibujaríamos el árbol sobre un papel. Así, cada nodo del árbol está representado por un bloque corto de celdas de memoria contiguas, como el formato mostrado en la Figura 8.15.

Almacenar el árbol en memoria implica encontrar sendos bloques de celdas de memoria vacías, para albergar los nodos y enlazar esos nodos entre sí de acuerdo con la estructura de árbol deseada. Cada puntero deberá configurarse para que apunte al hijo izquierdo o derecho del nodo pertinente, o bien se le asignará el valor NIL si no hay más nodos en esa dirección del árbol. (Esto significa que un nodo terminal se caracteriza porque sus dos punteros tienen asignado el valor NIL.) Por último, reservamos una posición de memoria especial, denominada **puntero raíz**, donde almacenaremos la dirección del nodo raíz. Es este puntero raíz el que proporciona el acceso inicial al árbol.

En la Figura 8.16 se presenta un ejemplo de este sistema de almacenamiento enlazado; allí se muestra una estructura de árbol binario conceptual junto con una representación de cómo podría realmente aparecer ese árbol dentro de la memoria de la computadora. Observe que la disposición real de los nodos en la memoria principal puede ser bastante distinta de la disposición conceptual. Sin embargo, siguiendo el puntero raíz, podemos localizar el nodo raíz y luego recorrer cualquier ruta que deseemos por el árbol, siguiendo los punteros apropiados para saltar de nodo en nodo.

Una alternativa para almacenar un árbol binario como una estructura enlazada consiste en utilizar un único bloque contiguo de celdas de memoria para todo el árbol. Con esta técnica, almacenamos el nodo raíz del árbol en la primera celda del bloque. (Para simplificar, suponemos que cada nodo del árbol solo necesita una celda de memoria.). Después, almacenamos el hijo izquierdo del nodo raíz en la segunda celda, almacenamos el hijo derecho del nodo raíz en la tercera celda y, en general, continuamos almacenando los hijos izquierdo y derecho del nodo correspondiente a la celda n en las celdas $2n$ y $2n + 1$, respectivamente. Las celdas dentro del bloque que representen ubicaciones no utilizadas por el árbol se marcarán con un patrón de bits distintivo que indique la ausencia de datos. Utilizando esta técnica, el mismo árbol mostrado en la Figura 8.16 se almacenaría tal como se ilustra en la Figura 8.17. Observe que el sistema consiste básicamente en almacenar los nodos en forma de segmentos correspondientes a cada uno de los niveles del árbol, uno a continuación de otro. Es decir, la primera entrada del bloque será el nodo raíz, seguido de los hijos de la raíz, seguidos de los nietos de la raíz, y así sucesivamente.

A diferencia de la estructura enlazada que hemos descrito anteriormente, este sistema alternativo de almacenamiento proporciona un método eficiente para localizar el padre o el hermano de cualquier nodo. La ubicación del padre

Figura 8.15 La estructura de un nodo en un árbol binario.

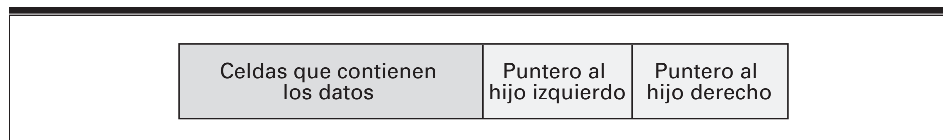


Figura 8.16 Organización conceptual y real de un árbol binario utilizando un sistema de almacenamiento enlazado.

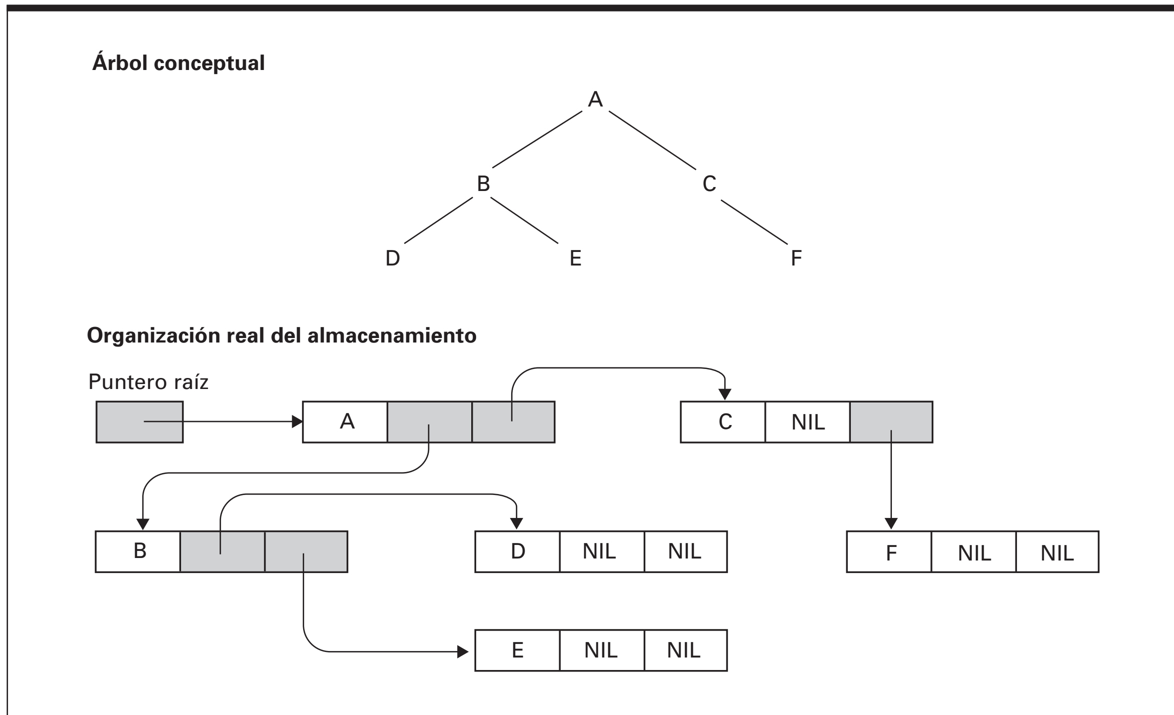
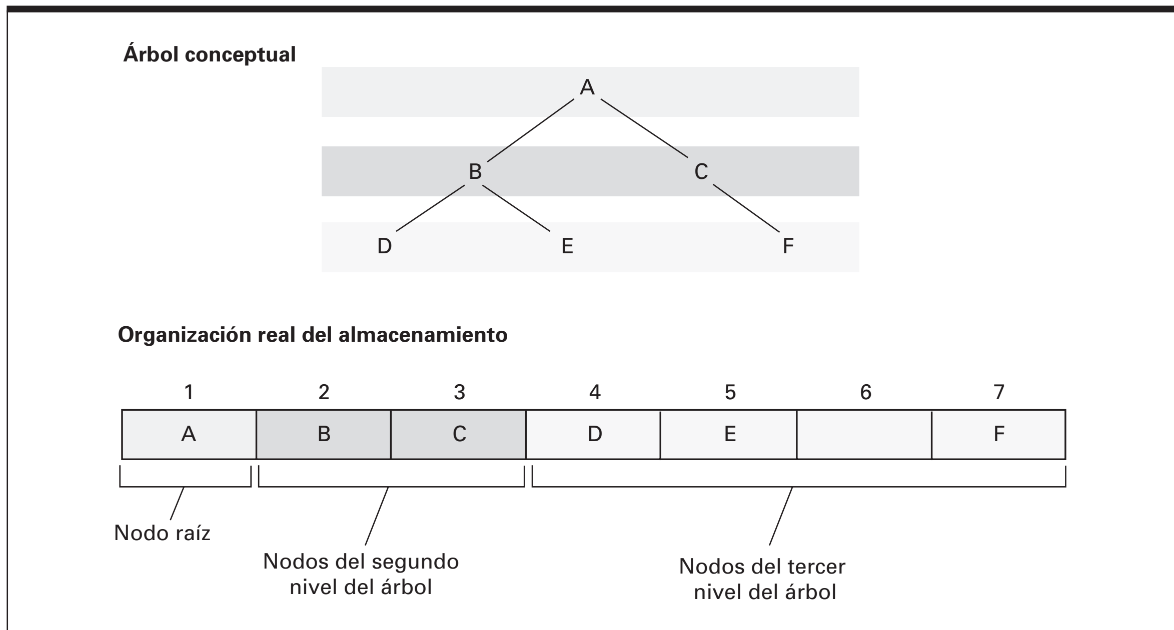


Figura 8.17 Un árbol almacenado sin punteros.



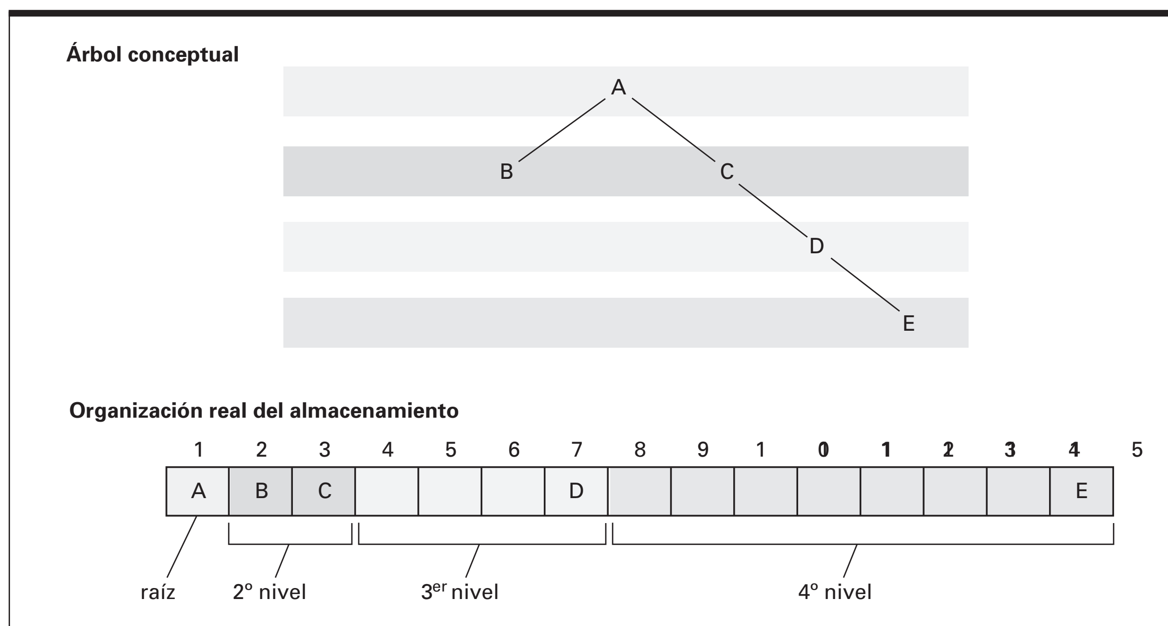
de un nodo puede encontrarse dividiendo por 2 la posición del nodo dentro del bloque y descartando el resto (el padre del nodo situado en la posición 7 sería el nodo de la posición 3). La ubicación del hermano de un nodo puede encontrarse sumando 1 a la ubicación del nodo si este está situado en una posición par o restando 1 de la posición de un nodo si este está situado en una posición impar. Por ejemplo, el hermano del nodo situado en la posición 4 será el nodo situado en la posición 5, mientras que el hermano del nodo que está en la posición 3 será el nodo situado en la posición 2. Además, este sistema de almacenamiento hace un uso muy eficiente del espacio cuando el árbol binario está más o menos equilibrado (en el sentido de que ambos subárboles del árbol raíz tengan la misma profundidad) y más o menos lleno (en el sentido de que no tengan ramas muy largas y finas). Sin embargo, para árboles que no cumplan estas condiciones, el sistema puede ser bastante ineficiente, como se ilustra en la Figura 8.18.

Manipulación de estructuras de datos

Hemos visto que la forma en que se almacenan realmente en la memoria de una computadora las estructuras de datos no coincide con la estructura conceptual, tal como el usuario se la representa mentalmente. Una matriz bidimensional no se almacena en realidad como un bloque rectangular bidimensional y una lista o un árbol pueden estar en la práctica compuestos por pequeños fragmentos dispersos por un área de memoria de gran tamaño.

Por tanto, para que el usuario pueda acceder a los datos de manera abstracta, debemos ocultar a sus ojos la complejidad del sistema de almacenamiento real utilizado. Esto significa que las instrucciones proporcionadas por el

Figura 8.18 Un árbol disperso y no equilibrado, mostrado en forma conceptual y en la forma en que se almacenaría sin utilizar punteros.



usuario (y enunciadas en términos de abstracción) deben convertirse en pasos que sean apropiados para el sistema de almacenamiento real empleado. En el caso de arrays, ya hemos visto cómo puede hacerse utilizando un polinomio de dirección para convertir los índices de fila y columna en direcciones de celdas de memoria. En particular, hemos visto cómo la sentencia

```
Ventas[3, 4] ← 5;
```

escrita por un programador que está pensando en términos de un array abstracto puede convertirse en una serie de pasos que realizan las modificaciones correctas en la memoria principal. De la misma forma, hemos visto cómo sentencias del tipo

```
Empleado.Edad ← 22;
```

que hacen referencia a una estructura abstracta, pueden traducirse en una serie de acciones apropiadas dependiendo de cómo esté realmente almacenada la estructura.

En el caso de listas, colas y árboles, las instrucciones expresadas en términos de la estructura abstracta suelen convertirse en las acciones apropiadas por medio de procedimientos que realizan la tarea deseada, al mismo tiempo que ocultan al usuario los detalles del sistema de almacenamiento subyacente. Por ejemplo, si proporcionamos el procedimiento `insertar` para introducir nuevas entradas en una lista enlazada, entonces podríamos insertar a Jaime García en la lista de estudiantes matriculados en el curso de Física Avanzada, ejecutando una llamada a procedimiento tal como

```
insertar("García, Jaime", FisicaAvanzada)
```

Observe que la llamada a procedimiento se expresa completamente en términos de la estructura abstracta (la forma en la que la lista se implemente en la práctica queda oculta a ojos del programador).

Veamos un ejemplo más detallado: la Figura 8.19 presenta un procedimiento llamado `imprimirLista` para imprimir una lista enlazada de nombres. Este procedimiento asume que hay un puntero inicial apuntando a la primera entrada de la lista y que cada entrada de la lista está formada por dos fragmentos: un nombre y un puntero a la siguiente entrada. Una vez escrito este procedimiento, puede utilizarse como una herramienta para imprimir una lista enlazada, sin preocuparnos de los pasos necesarios en la práctica para imprimir la lista. Por ejemplo, para obtener una lista impresa de los alumnos de la clase de Economía, un usuario solo tendría que hacer la llamada a procedimiento

```
imprimirLista(ListaClaseEconomia)
```

Figura 8.19 Procedimiento para imprimir una lista enlazada.

```
procedure ImprimirLista (Lista)
PunteroActual ← puntero inicial de Lista.
while (PunteroActual no es NIL) do
  (Imprimir el nombre de la entrada a la que apunta PunteroActual;
   Obtener el valor contenido en la celda de puntero de la entrada de Lista a
    la que apunta PunteroActual, y asignar dicho valor a PunteroActual.)
```

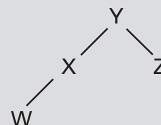
para obtener los resultados deseados. Además, si posteriormente decidimos modificar la forma en la que está almacenada realmente la lista, entonces solo tendríamos que cambiar las acciones internas del procedimiento `imprimirLista`; el usuario continuaría solicitando la impresión de la misma con la misma llamada a procedimiento que antes.

Cuestiones y ejercicios

1. Indique cómo se almacenaría la siguiente matriz en la memoria principal, si se almacena con ordenación por filas.

5	3	7
4	2	8
1	9	6

2. Proporcione una fórmula para localizar la entrada correspondiente a la ***i***-ésima fila y la ***j***-ésima columna de una matriz bidimensional si se almacena con ordenación por columnas en lugar de con ordenación por filas.
3. En los lenguajes de programación C, C++, Java y C#, los índices de las matrices comienzan en 0 en lugar de en 1. Por tanto, la entrada correspondiente a la cuarta columna de la primera fila de una matriz de nombre `Matriz` se indicaría mediante referencia `Matriz[0][3]`. En este caso, ¿qué polinomio de dirección utilizaría el traductor para convertir las referencias de la forma `Matriz[i][j]` a direcciones de memoria?
4. ¿Qué condición indica que una lista enlazada está vacía?
5. Modifique el procedimiento de la Figura 8.19 para que deje de imprimir una vez impreso un nombre determinado.
6. Basándose en la técnica expuesta en esta sección para la implementación de una pila en un bloque de celdas contiguo, ¿qué condición indica que la pila está vacía?
7. Describa cómo puede implementarse una pila en un lenguaje de alto nivel en términos de una matriz unidimensional.
8. Cuando se implementa una cola de forma circular como se ha descrito en esta sección, ¿cuál es la relación entre los punteros inicial y final cuando la cola está vacía? ¿Y cuando la cola está llena? ¿Cómo puede detectarse si una cola está vacía o llena?
9. Dibuje un diagrama que represente cómo aparecería en la memoria el árbol mostrado a continuación, si se almacena utilizando los punteros a los hijos izquierdo y derecho, como se ha descrito en esta sección. Después, dibuje otro diagrama que muestre cómo aparecería el árbol en un bloque de almacenamiento contiguo, empleando el sistema de almacenamiento alternativo descrito en esta sección.



8.4 Un pequeño caso de estudio

Vamos a considerar la tarea de almacenar una lista de nombres en orden alfabético. Vamos a asumir que las operaciones que hay que realizar con esta lista son las siguientes:

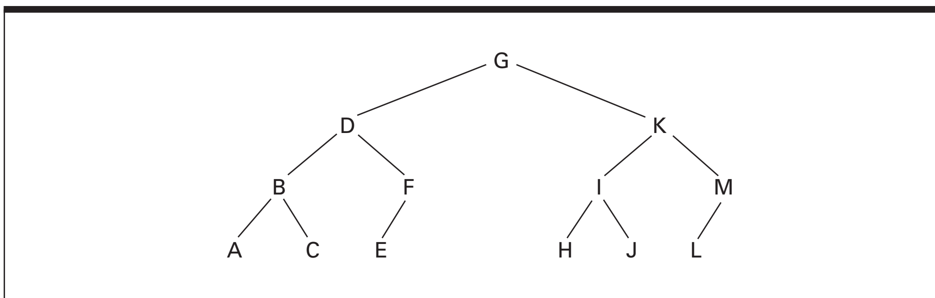
buscar si existe una cierta entrada,
imprimir la lista en orden alfabético e
insertar una nueva entrada.

Nuestro objetivo es desarrollar un sistema de almacenamiento y un conjunto de procedimientos para realizar estas operaciones, produciendo así una abstracción completa.

Comenzamos considerando las opciones de almacenamiento de la lista. Si la almacenamos con el modelo de lista enlazada, necesitaremos buscar en la lista de manera secuencial, un proceso que, como ya hemos explicado en el Capítulo 5, podría ser muy ineficiente si la lista llegara a tener una gran longitud. Por tanto, buscaremos una implementación que nos permita emplear el algoritmo de búsqueda binaria (Sección 5.5) como procedimiento de búsqueda. Para aplicar este algoritmo, nuestro sistema de almacenamiento debe permitirnos encontrar la entrada intermedia en una serie de porciones de la lista sucesivamente más pequeñas. La solución que adoptaremos consistirá en almacenar la lista en forma de árbol binario. Haremos que la entrada intermedia de la lista sea el nodo raíz. Después, haremos que la entrada intermedia de la primera mitad restante de la lista sea el hijo izquierdo del nodo raíz y que la entrada intermedia de la segunda mitad restante de la lista sea el hijo derecho del nodo raíz. Las entradas intermedias de cada uno de los cuartos de lista restantes serán los hijos de los hijos del nodo raíz, y así sucesivamente. Por ejemplo, el árbol de la Figura 8.20 representa la lista de letras A, B, C, D, E, F, G, H, I, J, K, L y M. (Cuando la parte de la lista en cuestión contenga un número par de entradas, consideraremos que la entrada intermedia es la mayor de las dos entradas centrales.)

Para efectuar búsquedas en la lista teniéndola almacenada de esta manera, compararemos el valor buscado con el nodo raíz. Si los dos son iguales, la búsqueda habrá tenido éxito. Si son distintos, nos moveremos al hijo izquierdo o al hijo derecho de la raíz dependiendo de si el valor buscado es menor o mayor que la raíz, respectivamente. Al desplazarnos a uno de los hijos, ya tenemos localizada la entrada intermedia de aquella parte de la lista que necesitamos

Figura 8.20 Las letras A hasta M dispuestas en un árbol ordenado.



para continuar la búsqueda. Este proceso de comparación y de desplazamiento a uno de los hijos continuará hasta que encontremos el valor buscado (lo que querrá decir que la búsqueda ha tenido éxito) o hasta que alcancemos un puntero NIL sin haber encontrado el valor buscado (lo que significará que la búsqueda ha fallado).

La Figura 8.21 muestra cómo podría expresarse este proceso de búsqueda en el caso de una estructura de árbol enlazado. Observe que este procedimiento es tan solo un refinamiento del procedimiento mostrado en la Figura 5.14, que es la forma en la que originalmente habíamos implementado la búsqueda binaria. Las diferencias son más bien de carácter cosmético. En lugar de expresar el algoritmo en términos de búsquedas realizadas en segmentos sucesivamente más pequeños de la lista, ahora expresamos el algoritmo en términos de búsqueda en subárboles sucesivamente más pequeños (Figura 8.22).

Habiendo almacenado nuestra "lista" en forma de árbol binario, podríamos pensar que el proceso de imprimir la lista en orden alfabético puede ser ahora difícil. Sin embargo, para ello, simplemente necesitamos imprimir el subárbol izquierdo en orden alfabético, imprimir el nodo raíz y luego imprimir el subárbol derecho en orden alfabético (Figura 8.23). Después de todo, el subárbol izquierdo contiene todos los elementos que son menores que el nodo raíz, mientras que el subárbol derecho contiene todos los elementos que son mayores que el nodo raíz. Podemos esbozar este proceso de la siguiente forma:

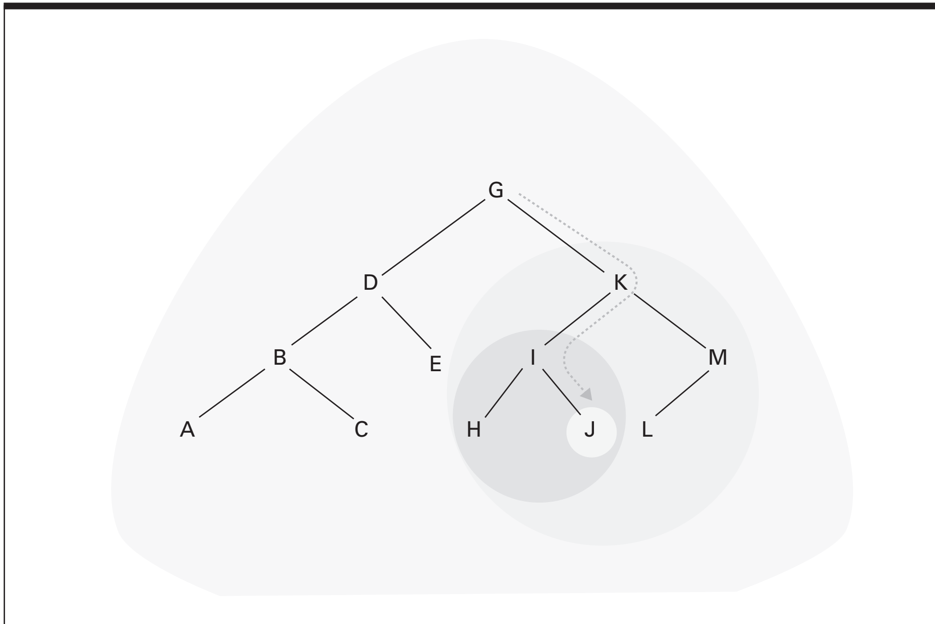
```
if (árbol no vacío)
then (imprimir el subárbol izquierdo en orden alfabético;
      imprimir el nodo raíz;
      imprimir el subárbol derecho en orden alfabético)
```

Figura 8.21 La búsqueda binaria, tal como se presentaría si implementáramos la lista en forma de árbol binario enlazado.

```
procedure Buscar(Arbol, ValorBuscado)

if (puntero raíz de Arbol = NIL)
  then
    (declarar que la búsqueda ha fallado)
  else
    (ejecutar el bloque de instrucciones que
     esté asociado con el caso apropiado)
     caso 1: ValorBuscado = valor del nodo raíz
            (Informar de que la búsqueda ha tenido éxito)
     caso 2: ValorBuscado < valor del nodo raíz
            (Aplicar el procedimiento Buscar para ver si
              ValorBuscado está en el subárbol identificado
                por el puntero al hijo izquierdo del nodo raíz
                e informar del resultado de esa búsqueda)
     caso 3: ValorBuscado > valor del nodo raíz
            (Aplicar el procedimiento Buscar para ver si
                ValorBuscado está en el subárbol identificado
                por el puntero al hijo derecho del nodo raíz
                e informar del resultado de esa búsqueda)
    ) end if
```

Figura 8.22 Los árboles sucesivamente más pequeños que tomará en consideración el procedimiento de la Figura 8.21 al buscar la letra J.

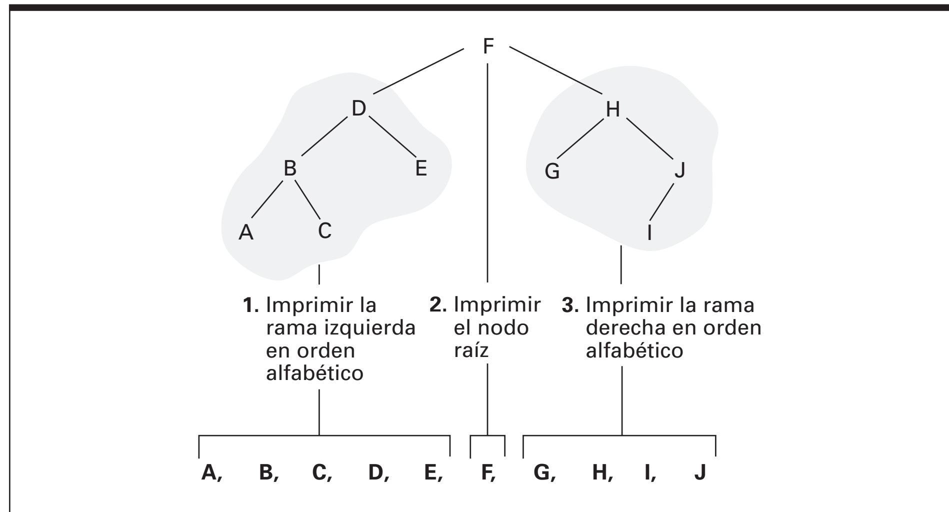


Recolección de basura

A medida que crecen y se contraen las estructuras de datos dinámicas, vamos utilizando y liberando espacio de almacenamiento. El proceso de reclamar el espacio de almacenamiento no usado para poder utilizarlo en el futuro se conoce con el nombre de **recolección de basura**. La recolección de basura es necesaria en diferentes escenarios. El gestor de memoria de un sistema operativo debe realizar la recolección de basura a medida que va asignando y liberando espacio de memoria. El administrador de archivos realiza una recolección de basura a medida que se almacenan y borran archivos en el almacenamiento masivo de la máquina. Además, cualquier proceso que se esté ejecutando bajo control del despachador puede necesitar realizar una recolección de basura dentro de su propio espacio de memoria asignado.

La recolección de basura presenta ciertos problemas sutiles. En el caso de estructuras enlazadas, cada vez que se cambia un puntero a un elemento de datos, el recolector de basura debe decidir si hay que reclamar el espacio de almacenamiento al que apuntaba originalmente el puntero. El problema se vuelve especialmente complejo en las estructuras de datos entremezcladas, en las que pueden existir múltiples cadenas de punteros. La utilización de rutinas de recolección de basura inadecuadas pueden conducir a la pérdida de datos o a un uso ineficiente del espacio de almacenamiento. Por ejemplo, si la recolección de basura no consiguiera reclamar adecuadamente el espacio de almacenamiento, el espacio disponibles se iría reduciendo lentamente, lo cual es un fenómeno que se conoce con el nombre de **pérdidas de memoria**.

Figura 8.23 Impresión de un árbol de búsqueda en orden alfabético.



Este esbozo incluye las tareas de imprimir el subárbol izquierdo y el subárbol derecho en orden alfabético, siendo ambas tareas básicamente versiones reducidas de la tarea original. Es decir, resolver el problema de imprimir un árbol, implica la tarea más pequeña de imprimir un subárbol, lo que sugiere el uso de una técnica recursiva para resolver nuestro problema de impresión del árbol.

Teniendo esto en cuenta, podemos ampliar nuestra idea inicial y construir un procedimiento completo en pseudocódigo para imprimir el árbol, como se muestra en la Figura 8.24. Hemos asignado a la rutina el nombre `ImprimirArbol` y luego hemos solicitado los servicios de `ImprimirArbol` para imprimir los subárboles izquierdo y derecho. Observe que la condición de terminación del proceso recursivo (que es que se alcance un subárbol vacío) siempre terminará por producirse, porque cada activación sucesiva de la rutina opera sobre un árbol más pequeño que el que ha causado la activación.

La tarea de insertar una entrada nueva en el árbol también es más fácil de lo que podría parecer en un principio. La intuición podría hacernos creer que las inserciones obligan a cortar el árbol para hacer sitio para otras entradas, pero en realidad el nodo que se está añadiendo siempre puede conectarse como una nueva hoja, independientemente de cuál sea su valor. Para encontrar el lugar apropiado en el que insertar una nueva entrada, vamos descen-

Figura 8.24 Un procedimiento para imprimir los datos en un árbol binario.

```
procedure ImprimirArbol (Arbol)

if (Arbol no vacío)
   then (Aplicar el procedimiento ImprimirArbol al árbol que
             aparezca como la rama izquierda de Arbol;
         Imprimir el nodo raíz de Arbol;
         Aplicar el procedimiento ImprimirArbol al árbol que
             aparezca como la rama derecha de Arbol)
```

diendo por el árbol siguiendo la misma ruta que habría que seguir si estuviéramos buscando ese valor concreto. Puesto que la entrada no se encuentra en el árbol, nuestra búsqueda terminará por llevarnos a un puntero NIL. En este punto, habremos localizado la ubicación apropiada para el nuevo nodo (Figura 8.25). De hecho, esta es la ubicación a la que conduciría una operación de búsqueda de esa nueva entrada.

En la Figura 8.26 se muestra un procedimiento que implementa este proceso para el caso de una estructura de árbol enlazado. El procedimiento busca en el árbol el valor que se quiere insertar (denominado `NuevoValor`) y luego inserta un nuevo nodo hoja que contiene `NuevoValor` en la ubicación apropiada. Observe que si durante esa búsqueda se encontrara en el árbol la entrada que se pretende insertar, la inserción no se llevaría a cabo.

Podemos concluir que un paquete software compuesto por una estructura de árbol binario enlazado y por nuestros procedimientos para buscar, imprimir e insertar proporciona un paquete completo que una hipotética aplicación podría utilizar como herramienta abstracta. De hecho, si se implementa adecuadamente, este paquete puede utilizarse sin preocuparse de cuál es la estructura subyacente de almacenamiento. Utilizando los procedimientos del paquete, el usuario podría pensar en una lista de nombres almacenada en orden alfabético, aunque la realidad fuera que las entradas de la "lista" están dispersas entre una serie de bloques de celdas de memoria que están enlazados en forma de árbol binario.

Figura 8.25 Inserción de la entrada M en la lista B, E, G, H, J, K, N, P almacenada como un árbol.

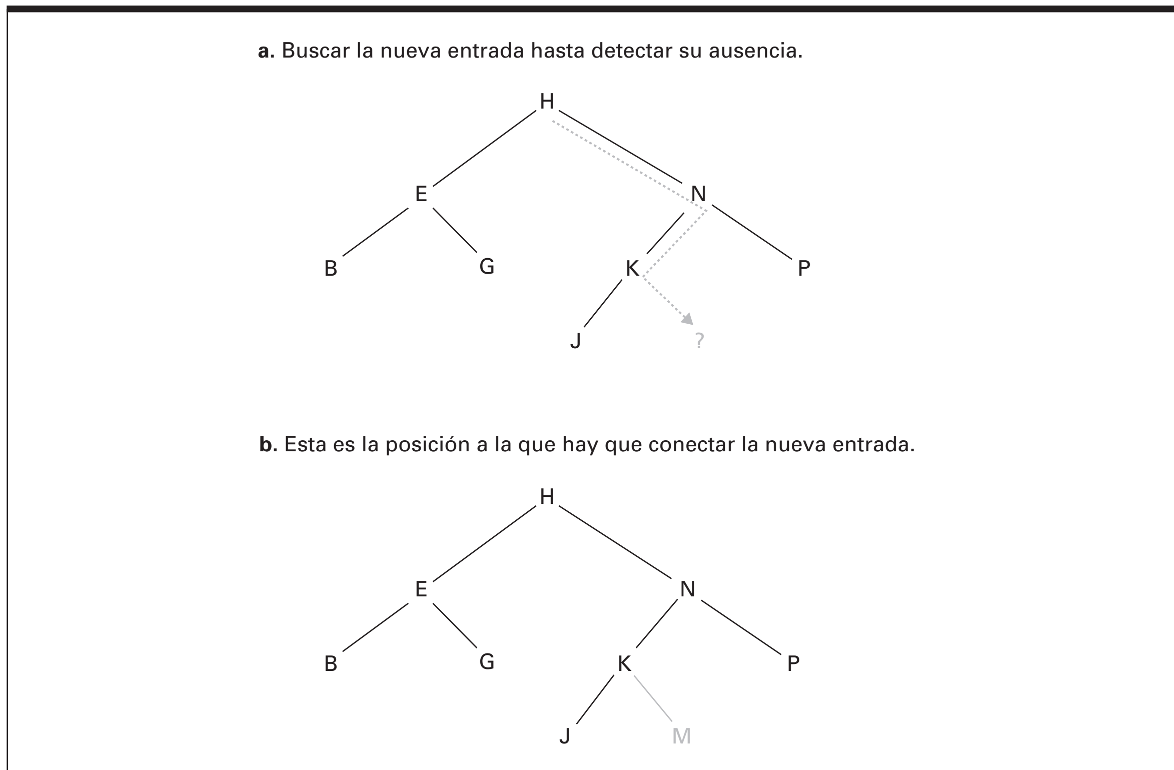


Figura 8.26 Un procedimiento para insertar una nueva entrada en un árbol binario.

```
procedure Insertar (Arbol, NuevoValor)

if (puntero raíz de Arbol = NIL)
  (hacer que el puntero raíz apunte a una nueva hoja
       que contenga NuevoValor)
  else (ejecutar de entre los bloques de instrucciones siguientes
       el que esté asociado con el caso apropiado)
        caso 1: NuevoValor = valor del nodo raíz
              (No hacer nada)
        caso 2: NuevoValor < valor del nodo raíz
              (if (puntero hijo izquierdo del nodo raíz = NIL)
                     then (hacer que ese puntero apunte a un nuevo
                             nodo hoja que contenga NuevoValor)
                     else (aplicar el procedimiento Insertar para insertar
                             NuevoValor en el subárbol identificado
                             por el puntero hijo izquierdo)
        caso 3: NuevoValor > valor del nodo raíz
              (if (puntero hijo derecho del nodo raíz = NIL)
                     then (hacer que ese puntero apunte a un nuevo
                             nodo hoja que contenga NuevoValor)
                     else (aplicar el procedimiento Insertar para insertar
                             NuevoValor en el subárbol identificado
                             por el puntero hijo derecho)
     ) end if
```

Cuestiones y ejercicios

1. Dibuje un árbol binario que pueda utilizarse para almacenar la lista R, S, T, U, V, W, X, Y y Z y poder posteriormente realizar búsquedas.
2. Indique la ruta recorrida por el algoritmo de búsqueda binaria de la Figura 8.21 si lo aplicamos al árbol de la Figura 8.20 para buscar la entrada J. ¿Y en el caso de la entrada P?
3. Dibuje un diagrama que represente el estado de las activaciones del algoritmo recursivo de impresión del árbol de la Figura 8.24, en el instante en que se imprime el nodo K dentro del árbol ordenado de la Figura 8.20.
4. Describa cómo podría utilizarse una estructura de árbol en la que cada nodo tuviera hasta 26 hijos para codificar la ortografía correcta de las palabras en el idioma inglés.

8.5 Tipos de datos personalizados

En el Capítulo 6 hemos presentado el concepto de tipo de datos y hemos hablado de algunos tipos elementales como los enteros, los reales, los caracteres y los booleanos. Estos tipos de datos se ofrecen en la mayoría de los lenguajes de programación como tipos de datos primitivos. En esta sección vamos a ver de qué manera pueden los programadores definir sus propios tipos de datos, para ajustarse mejor a las neecsidades de una aplicación concreta.

Tipos de datos definidos por el usuario

Expresar un algoritmo suele ser más fácil si hay disponibles tipos de datos distintos que los proporcionados como primitivos en los lenguajes de programación. Por esta razón, muchos lenguajes de programación modernos permiten a los programadores definir tipos de datos adicionales, utilizando los tipos primitivos como componentes básicos. Los ejemplos más elementales de estos tipos de datos "caseros" son los que se conocen como **tipos de datos definidos por el usuario**, que son básicamente conglomerados de tipos primitivos agrupados bajo un nombre común.

Para explicar este concepto, suponga que deseamos desarrollar un programa que tenga numerosas variables, cada una con la misma estructura compuesta por un nombre, una edad y una categoría. Una solución sería definir cada variable por separado como una estructura (Sección 6.2). Sin embargo, otra solución mejor sería definir esa estructura heterogénea como un nuevo tipo de dato (definido por el usuario) y luego usar ese nuevo tipo como si fuera una primitiva.

Para implementar esta idea, podríamos utilizar una instrucción de pseudocódigo de la forma

```
define type TipoEmpleado to be
{char Nombre[25];
 int  Edad;
 real Categoria;
}
```

para definir un nuevo tipo denominado `TipoEmpleado`, que estará compuesto por una estructura heterogéna que contiene los componentes denominados `Nombre` (de tipo carácter), `Edad` (de tipo entero) y `Categoria` (de tipo real). Después, podemos utilizar este nuevo tipo de dato para declarar variables de la misma manera que usamos los tipos de datos primitivos. Es decir, al igual que la mayoría de los lenguajes de programación permiten declarar la variable `x` como entero mediante la sentencia

```
int x;
```

podríamos también declarar la variable `Empleado1` como de tipo `TipoEmpleado` mediante la sentencia

```
TipoEmpleado Empleado1;
```

Entonces, posteriormente en el programa, la variable `Empleado1` haría referencia a un bloque completo de celdas de memoria que contiene el nombre, la edad y la categoría de un empleado. Para hacer referencia a los elementos individuales dentro del bloque podríamos emplear sentencias como `Empleado1.Nombre` y `Empleado1.Edad`. Así, una sentencia del tipo

```
Empleado1.Edad ← 26;
```

podría emplearse para asignar el valor 26 al componente `Edad` dentro del bloque conocido como `Empleado1`. Asimismo, la sentencia

```
TipoEmpleado DirectorDist, RepVentas1, RepVentas2;
```

podría utilizarse para declarar las tres variables `DirectorDist`, `RepVentas1` y `RepVentas2` como de tipo `TipoEmpleado`, al igual que usaríamos una sentencia de la forma

```
real Manga, Cintura, Cuello;
```

para declarar las variables `Manga`, `Cintura` y `Cuello` como del tipo primitivo `real`.

Es importante distinguir entre un tipo de dato definido por el usuario y un elemento concreto de dicho tipo. A este último se le suele denominar **instancia** de ese tipo de dato. Un tipo de dato definido por el usuario es básicamente una plantilla que se utiliza para construir instancias de este tipo. Describe las propiedades que tienen todas las instancias de ese tipo, pero no declara por sí mismo ninguna instancia de ese tipo (al igual que un molde para galletas es una plantilla a partir de la cual pueden hacerse galletas, pero ella misma no es una galleta). En el ejemplo anterior se utilizaba el tipo de dato definido por el usuario `TipoEmpleado` para construir tres instancias de dicho tipo, conocidas con los nombres de `DirectorDist`, `RepVentas1` y `RepVentas2`.

Tipos abstractos de datos

Aunque el concepto de tipo de datos definido por el usuario es muy útil, no termina de resolver completamente la necesidad de creación de nuevos tipos de datos. Un tipo de datos completo está compuesto por dos partes: (1) un sistema de almacenamiento predeterminado (como un sistema en complemento a dos en el caso del tipo entero y un sistema de punto flotante en el caso del tipo real) y (2) una colección de operaciones predefinidas (como la suma y la resta). En particular, los tipos de datos primitivos en un lenguaje de programación están asociados con una serie de operaciones primitivas. Si un programador declara una variable como de tipo primitivo, puede comenzar a aplicar de inmediato operaciones primitivas a esa variable, sin necesidad de efectuar definiciones adicionales.

Sin embargo, los tipos de datos definidos por el usuario tradicionales simplemente permiten a los programadores definir nuevos sistemas de almacenamiento, no proporcionan operaciones que aplicar a los datos que tengan esa estructura. Para clarificar este aspecto, suponga que queremos crear y utilizar varias pilas de valores enteros dentro de un programa. Nuestra solución podría ser implementar cada pila como una matriz de 20 valores enteros. La entrada del fondo de la pila se colocaría (introduciría) en la primera posición de la matriz y las entradas adicionales de la pila se colocarían (introducirían) en entradas sucesivamente mayores de esa matriz (véase la Cuestión/Ejercicio 7 de la Sección 8.3). Utilizaríamos una variable adicional de tipo entero como puntero de pila, la cual almacenaría el índice de la entrada de la matriz en la que habría que introducir la siguiente entrada de la pila. Así, cada pila estaría compuesta por una matriz que contendría a la propia pila y un entero que desempeñaría el papel de puntero de pila.

Para implementar esta solución, primero declararíamos un tipo de dato definido por el usuario denominado `TipoPila` con una sentencia de la forma

```
define type TipoPila to be
{int EntradasPila[20];
 int PunteroPila = 0;
}
```

(Observe que, según lo habitual en lenguajes tales como C, C++, C# y Java, estamos suponiendo que los índices para la matriz EntradasPila va de 0 a 19, por lo que hemos inicializado PunteroPila con el valor 0.) Habiendo hecho esta declaración, podemos luego declarar pilas denominadas PilaUno, PilaDos y PilaTres por medio de la sentencia

```
TipoPila PilaUno, PilaDos, PilaTres;
```

Llegados a este punto, cada una de las variables PilaUno, PilaDos y PilaTres hará referencia a un bloque distinto de celdas de memoria utilizado para implementar una pila individual.

¿Pero qué pasa si ahora queremos introducir el valor 25 en PilaUno? Nos gustaría no tener que ocuparnos de los detalles de la estructura de matriz subyacente a la implementación de la pila y emplear simplemente la pila como una herramienta abstracta, quizá usando una llamada a procedimiento similar a

```
insertar(25, PilaUno)
```

Pero dicha instrucción no estará disponible a menos que también definamos un procedimiento apropiado denominado insertar. Otras operaciones que nos gustaría realizar sobre las variables de tipo TipoPila incluirían la extracción de entradas de la pila, ver si la pila está vacía o está llena, operaciones que precisarían la defnción de procedimientos adicionales. En resumen, nuestra definición del tipo de datos TipoPila no incluye todas las propiedades que nos gustaría tener asociadas con ese tipo.

Podríamos resolver este problema ampliando nuestra sentencia define type para incluir procedimientos, así como descripciones de datos. Por ejemplo, podríamos escribir

```
define type TipoPila to be
{int EntradasPila[20];
 int PunteroPila = 0;
 procedure insertar(valor)
  {EntradasPila[ PunteroPila] ← valor;
   PunteroPila ← PunteroPila + 1;
  }
 procedure extraer . . .
}
```

que pretende significar que el tipo TipoPila está asociado con unas variables denominadas EntradasPila y PunteroPila y con unos procedimientos denominados insertar y extraer. (En aras de la simplicidad, hemos incluido una versión bastante simplona del procedimiento insertar. En realidad, el procedimiento debería asegurarse de que la pila no esté llena antes de intentar insertar una entrada adicional.)

Con esta definición ampliada del tipo TipoPila, podríamos declarar que PilaUno, PilaDos y PilaTres son pilas mediante la sentencia

```
TipoPila PilaUno, PilaDos, PilaTres;
```

Entonces podríamos insertar entradas en estas pilas con sentencias como

```
PilaUno.insertar(25);
```

que significa que hay que ejecutar el procedimiento `insertar` asociado con `PilaUno` utilizando el valor 25 como parámetro real.

Los tipos de datos definidos por el usuario que incluyen también definiciones de operaciones se denominan **tipos abstractos de datos**. Así, por oposición a los tipos de datos definidos por el usuario que son más elementales, los tipos abstractos de datos son tipos de datos completos y su aparición en lenguajes tales como Ada en la década de 1980 representó un importante paso hacia adelante en el diseño de los lenguajes de programación. Actualmente, los lenguajes orientados a objetos proporcionan versiones ampliadas de los tipos de datos abstractos. Esas versiones ampliadas se denominan clases, como veremos en la siguiente sección.

Cuestiones y ejercicios

1. ¿Cuál es la diferencia entre un tipo de datos y una instancia de ese tipo?
2. ¿Cuál es la diferencia entre un tipo de datos definido por el usuario y un tipo abstracto de datos?
3. Describa un tipo abstracto de datos para implementar una lista.
4. Describa un tipo abstracto de datos para la implementación de cuentas corrientes en un banco.

8.6 Clases y objetos

Como hemos visto en el Capítulo 6, el paradigma orientado a objetos conduce a sistemas compuestos por unidades denominadas objetos, que interactúan entre sí para llevar a cabo una serie de tareas. Cada objeto es una entidad que responde a mensajes recibidos de otros objetos. Los objetos se describen mediante plantillas que se conocen con el nombre de clases.

En muchos aspectos, estas clases son en realidad descripciones de tipos abstractos de datos (cuyas instancias se denominan objetos). De hecho, las instrucciones utilizadas para definir clases en lenguajes de programación orientados a objetos muy populares son enormemente similares a la instrucción `define type` presentada en la sección anterior. Por ejemplo, la Figura 8.27 muestra cómo puede definirse una clase de nombre `PilaDeEnteros` en los lenguajes Java y C#. (La definición equivalente de clase en C++ tiene la misma estructura, pero una sintaxis ligeramente diferente.) Observe la similitud entre esta clase y la instrucción `define type` que hemos usado en la sección anterior para describir el tipo abstracto de datos `TipoPila`. Describe que la clase/tipo contiene una matriz de enteros denominada `EntradasPila`, un entero que sirve para identificar la cima de la pila dentro de la matriz y que se denomina `PunteroPila` y una serie de procedimientos que sirven para manipular la pila.

Utilizando esta clase como plantilla, puede crearse un objeto de nombre `PilaUno` en un programa Java o C# mediante la sentencia

Figura 8.27 Una pila de enteros implementada en Java y C#.

```
class PilaDeEnteros
{private int[] EntradasPila = new int[20];
 private int PunteroPila = 0;


 public void insertar(int NuevaEntrada)
 {if (PunteroPila < 20)
    EntradasPila[PunteroPila++] = NuevaEntrada;
 }

 public int extraer ()
 {if (PunteroPila > 0) return EntradasPila[--PunteroPila];
  else return 0;
 }
}
```

```
PilaDeEnteros PilaUno = new PilaDeEnteros();
```

o en un programa C++ mediante la sentencia

```
PilaDeEnteros PilaUno();
```

Posteriormente en los programas, podemos insertar el valor 106 en `PilaUno` utilizando la sentencia

```
PilaUno.insertar(106);
```

o podemos extraer la entrada superior de `PilaUno` y colocarla en la variable `AntiguoValor` utilizando la sentencia

```
AntiguoValor = PilaUno.extraer();
```

Estas características son esencialmente las mismas que las asociadas con los tipos abstractos de datos. Sin embargo, existen diferencias entre los tipos abstractos de datos y las clases. Las clases son una extensión de los tipos abstractos de datos. Por ejemplo, como hemos visto en la Sección 6.5, los lenguajes orientados a objetos permiten que las clases hereden propiedades de otras clases

La librería STL

Las estructuras de datos de las que hemos hablado en este capítulo se han convertido en estructuras estándar de programación, tan estándar, de hecho, que muchos entornos de programación las tratan casi como si fueran primitivas. Podemos encontrar un ejemplo en el entorno de programación C++, que está ampliado con la librería STL (***Standard Template Library***, Librería estándar de plantillas). La STL contiene un conjunto de clases predefinidas que describen diversas estructuras de datos muy populares. En consecuencia, incorporando la STL en un programa C++, el programador queda liberado de describir estas estructuras en detalle. En lugar de ello, lo único que tiene que hacer es declarar los identificadores de los tipos apropiados, de la misma manera que declarábamos `PilaUno` como de tipo `PilaDeEnteros` en la Sección 8.6.

y también permiten que las clases contengan métodos especiales denominados constructores, que personalizan los objetos individuales en el momento de crearlos. Además, las clases se asocian normalmente con diversos grados de encapsulación (Sección 6.5), permitiendo proteger las propiedades internas de sus instancias frente a posibles atajos dañinos. Y, finalmente, una clase puede utilizarse como medio de agrupar procedimientos relacionados y puede, por tanto, estar constituida únicamente por definiciones de procedimientos. En este sentido, podríamos decir que una clase es un tipo abstracto más que un tipo abstracto de datos.

Podemos concluir que los conceptos de clases y de objetos constituyen otro paso en la evolución de las técnicas para representar abstracciones de datos en los programas. De hecho, es esta capacidad de definir y utilizar abstracciones de una forma conveniente lo que ha dotado de tanta popularidad al paradigma orientado a objetos.

Cuestiones y ejercicios

1. ¿En qué se parecen los tipos abstractos de datos y las clases? ¿En qué se diferencian?
2. ¿Cuál es la diferencia entre una clase y un objeto?
3. Describa una clase que podríamos utilizar como plantilla para construir objetos de tipo cola-de-enteros.

8.7 Punteros en el lenguaje máquina

En este capítulo hemos presentado los punteros y hemos mostrado cómo se utilizan a la hora de construir estructuras de datos. En esta sección vamos a considerar cómo se manipulan los punteros en lenguaje máquina.

Suponga que queremos escribir un programa en el lenguaje máquina descrito en el Apéndice C con el fin de extraer una entrada de una pila, como se describe en la Figura 8.12 y colocar dicha entrada en un registro de propósito general. En otras palabras, queremos cargar un registro con el contenido de la posición de memoria que contiene la entrada correspondiente a la cima de la pila. Nuestro lenguaje máquina proporciona dos instrucciones para cargar registros: una con el código de operación 2 y la otra con el código de operación 1. Recuerde que en el caso del código de operación 2, el campo de operando contiene el dato que hay que cargar, mientras que en el caso del código de operación 1, el campo de operando contiene la dirección del dato que hay que cargar.

No sabemos cuál va a ser el contenido, así que no podemos utilizar el código de operación 2 para conseguir nuestro objetivo. Además, tampoco podemos utilizar el código de operación 1, porque no sabemos cuál será la dirección. Después de todo, la dirección de la cima de la pila irá variando a medida que se ejecute el programa. Sin embargo, lo que sí conocemos es la dirección del puntero de pila. Es decir, conocemos la posición de la dirección del dato que queremos cargar. Lo que necesitamos, entonces, es un tercer código de operación

para cargar un registro, en el que el operando contenga la dirección de un puntero que apunte al dato que hay que cargar.

Para conseguir esto, ampliaremos el lenguaje del Apéndice C con el fin de incluir un código de operación D. Una instrucción con este código de operación tendría la forma DRXY, lo que significa que hay que cargar en el registro R el contenido de la posición de memoria cuya dirección se encuentra en la dirección XY (Figura 8.28). Así, si el puntero de pila se encuentra en la posición de memoria en la dirección AA, entonces la instrucción D5AA haría que el dato situado en la cima de la pila se cargue en el registro 5.

Sin embargo, esta instrucción no completa la operación de extracción del dato. También deberemos restar una unidad del puntero de pila para que ahora apunte a la nueva cima de la pila. Esto significa que, después de la instrucción de carga, nuestro programa en lenguaje máquina tendría que cargar el puntero de pila en un registro, restarle una unidad y volver a almacenar el resultado en memoria.

Utilizando uno de los registros como puntero de pila en lugar de usar una posición de memoria, podríamos eliminar la necesidad de mover el puntero de pila repetidamente entre los registros y la memoria. Pero esto implicaría que tendríamos que rediseñar la instrucción de carga para que esta espere encontrar el puntero en un registro en lugar de en la memoria principal. Por tanto, en lugar de la solución anterior, podríamos definir una instrucción con el código de operación D que tuviera la forma DR0S, lo que sigificaría que hay que cargar el registro R con el contenido de la celda de memoria a la que apunta el registro S (Figura 8.29). Entonces, podría efectuarse una operación completa de extracción ejecutando primero esta instrucción y luego otra instrucción (o instrucciones) para restar una unidad del valor almacenado en el registro S.

Observe que se necesita una instrucción similar para implementar una operación de inserción en la pila. Podemos por tanto ampliar aún más el lenguaje descrito en el Apéndice C, introduciendo el código de operación E, de

Figura 8.28 Nuestro primer intento de ampliar el lenguaje máquina del Apéndice C para poder utilizar punteros.

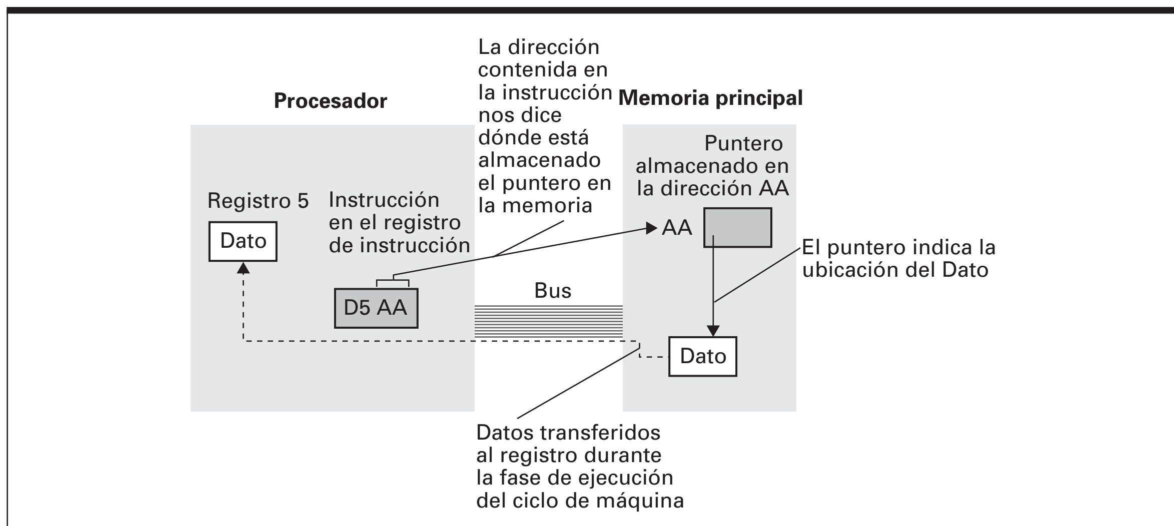
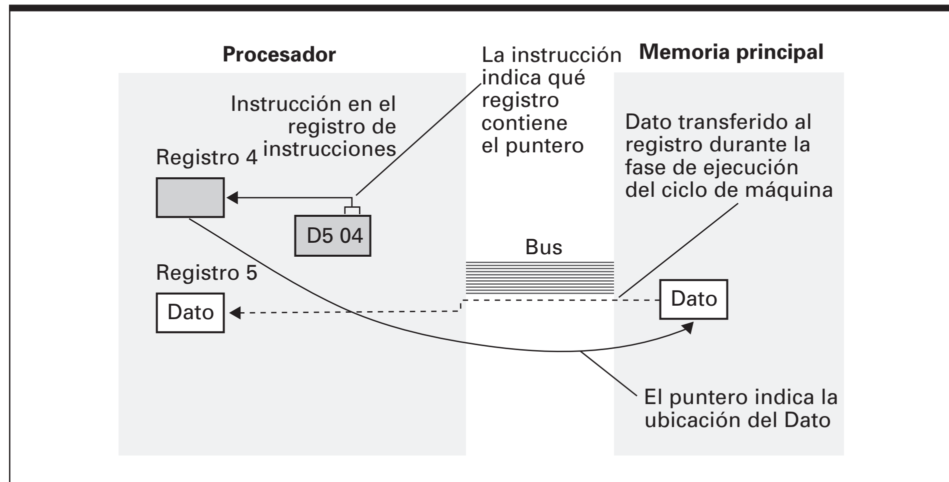


Figura 8.29 Carga de un registro a partir de una celda de memoria que se localiza por medio de un puntero almacenado en otro registro.



modo que una instrucción de la forma ER0S signifique que hay que almacenar el contenido del registro R en la celda de memoria a la que apunta el registro S. De nuevo, para completar la operación de inserción, esta instrucción debería ir seguida por otra instrucción (o instrucciones) que sumara uno al valor contenido en el registro S.

Estos nuevos códigos de operación D y E que hemos propuesto no solo ilustran cómo se diseñan los lenguajes máquina para manipular punteros, sino que también ilustran una técnica de direccionamiento que no estaba presente en el lenguaje máquina original. Tal como se presenta en el Apéndice C, el lenguaje máquina utiliza dos medios de identificar los datos implicados en una instrucción. El primero de ellos queda ilustrado por una instrucción cuyo código de operación sea 2. En ella, el campo de operando contiene explícitamente el dato necesario. Esta técnica se denomina **direccionamiento inmediato**. El segundo medio para identificar datos queda ilustrado por las instrucciones con los códigos de operación 1 y 3. En ellas, los campos de operando contienen la dirección del dato necesario. Esta técnica se denomina **direccionamiento directo**. Sin embargo, nuestros nuevos códigos de operación D y E propuestos ilustran otra forma más de identificar los datos. Los campos de operando de estas instrucciones contienen la dirección de la dirección de los datos. Esta técnica se denomina **direccionamiento indirecto**. Las tres técnicas de direccionamiento son bastante comunes en los lenguajes máquina actuales.

Cuestiones y ejercicios

1. Suponga que ampliamos el lenguaje máquina descrito en el Apéndice C como se ha sugerido al final de esta sección. Suponga también que el registro 8 contiene el patrón DB, que la celda de memoria en la posicion DB contiene el patrón CA y que la celda situada en la dirección CA con-

tiene el patrón A5. ¿Qué patrón de bits contendrá el registro 5 inmediatamente después de ejecutar cada una de las siguientes instrucciones?

a. 25A5
b. 15CA
c. D508

2. Utilizando las extensiones descritas al final de esta sección, escriba una rutina completa en lenguaje máquina para realizar una operación de extracción en una pila. Suponga que la pila está implementada como se muestra en la Figura 8.12, que el puntero de pila se encuentra en el registro F y que hay que extraer el valor situado en la cima de la pila y almacenarlo en el registro 5.
3. Utilizando las extensiones descritas al final de esta sección, escriba un programa para copiar el contenido de cinco celdas de memoria contiguas que comienzan en la dirección A0 a las cinco celdas que comienzan en la dirección B0. Suponga que el programa comienza en la dirección 00.
4. En este capítulo hemos presentado una instrucción de la forma DR0S. Suponga que ampliáramos esta instrucción a DRXS, con el siguiente significado "Cargar el registro R con el dato al que apunta el valor contenido en el registro S sumado con el valor X." Así, el puntero al dato se obtiene extrayendo el valor almacenado en el registro S y luego incrementando dicho valor en X. El valor del registro S no se ve alterado. (Si el registro F contuviera el valor 04, entonces la instrucción DE2F cargaría en el registro E el contenido de la celda de memoria situada en la dirección 06. El valor del registro F seguiría siendo 04.) ¿Qué ventajas tendría esta instrucción? ¿Qué pasaría con una instrucción de la forma DRTS, que significara "Cargar el registro R con el dato al que apunta el valor del registro S sumado al valor del registro T"?

Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. Dibuje sendos diagramas que muestren cómo aparecería en la memoria de la máquina la matriz que se muestra a continuación, si se la almacenara con ordenación por filas y con ordenación por columnas.

A	B	C	D
E	F	G	H
I	J	K	L

2. Suponga que almacenamos con ordenación por filas un array de seis filas y ocho columnas, comenzando en la dirección 20 (base diez). Si cada entrada de la matriz requiere una única celda de memoria, ¿cuál es la dirección correspondiente a la entrada de la cuarta columna de la tercera fila? ¿Y cuál sería la dirección si cada entrada requiriera dos celdas de memoria?

3. Repita el Problema 2 suponiendo que se utiliza ordenación por columnas en lugar de ordenación por filas.

4. ¿Qué complicaciones aparecen si se intenta implementar una lista dinámica utilizando una matriz unidimensional tradicional?

5. Describa un método para almacenar matrices tridimensionales. ¿Qué polinomio de dirección se utilizaría para localizar la entrada correspondiente al ***i***-ésimo plano, ***j***-ésima fila y ***k***-ésima columna?

6. Suponga que almacenamos la lista de letras A, B, C, E, F y G en un bloque contiguo de posiciones de memoria. ¿Qué actividades se requieren para insertar la letra D en la lista, suponiendo que haya que preservar el orden alfabético de la lista?

7. La siguiente tabla representa el contenido de algunas posiciones de la memoria principal de una computadora, junto con la dirección de cada celda representada. Observe que algunas de las celdas contienen letras del alfabeto y que cada una de esas celdas va seguida por una celda vacía. Coloque las direcciones necesarias en esas celdas vacías de manera que cada celda que contenga una letra forme, junto con la siguiente posición, una entrada de una lista enlazada en la que las letras aparezcan en orden alfabético. (Use el cero como puntero NIL.) ¿Qué dirección debería contener el puntero inicial?

Dirección	Contenido
11	C
12	
13	G
14	
15	E
16	
17	B
18	
19	U
20	
21	F
22	

8. La siguiente tabla representa una parte de una lista enlazada en la memoria principal de una computadora. Cada entrada de la lista está compuesta por dos celdas: la primera contiene una letra del alfabeto y la segunda contiene un puntero a la siguiente entrada de la lista. Modifique los punteros para que la letra N deje de pertenecer a la lista. Luego sustituya la letra N por la letra G y modifique los punteros para que la nueva letra aparezca en la lista en su lugar apropiado, en orden alfabético.

Dirección	Contenido
30	J
31	38
32	B
33	30
34	X
35	46
36	N
37	40
38	K
39	36
40	P
41	34

9. La siguiente tabla representa una lista enlazada utilizando el mismo formato que en los problemas anteriores. Si el puntero inicial contiene el valor 44, ¿cuál es el nombre representado por la lista? Modifique los punteros para que la lista contenga el nombre Jean.

Dirección	Contenido
40	N
41	46
42	I
43	40
44	J
45	50
46	E
47	00
48	M
49	42
50	A
51	40

10. ¿Cuál de las siguientes rutinas insertará correctamente `NuevaEntrada` inmediata-

mente después de la entrada denominada `EntradaAnterior` en una lista enlazada? ¿Qué es lo que falla en la otra rutina?

Rutina 1:

```
1. Copiar el valor contenido
   en el campo de puntero de
   EntradaAnterior en el campo de
   puntero de Nuevaentrada.
2. Modificar el valor contenido
   en el campo de puntero de
   EntradaAnterior introduciendo
   la dirección de Nuevaentrada.
```

Rutina 2:

```
1. Modificar el valor contenido
   en el campo de puntero de
   EntradaAnterior introduciendo
   la dirección de Nuevaentrada.
2. Copiar el valor del campo de
   puntero de EntradaAnterior en
   el campo de puntero de NuevaEn-
   trada.
```

11. Diseñe un procedimiento para concatenar dos listas enlazadas (es decir, para colocar una delante de la otra, con el fin de formar una única lista).

12. Diseñe un procedimiento para combinar dos listas contiguas ordenadas en una única lista ordenada contigua. ¿Qué sucede si las listas están enlazadas?

13. Diseñe un procedimiento para invertir el orden de una lista enlazada.

14. a. Diseñe un algoritmo para imprimir una lista enlazada en orden inverso, utilizando una pila como estructura auxiliar de almacenamiento.

b. Diseñe un procedimiento recursivo para realizar esta misma tarea sin hacer un uso explícito de una pila. ¿En qué forma seguirá habiendo una pila implicada en esa solución recursiva?

15. En ocasiones, a una única lista enlazada se le asignan dos ordenaciones diferentes, asociando dos punteros con cada entrada en lugar de uno. Rellene la tabla que se muestra a continuación de modo que, al seguir el primer puntero después de cada letra aparezca el nombre Carol, pero al seguir el segundo puntero después de cada letra nos encontremos las letras en orden alfabético. ¿Qué valor tendrá el puntero inicial de cada una de las dos listas representadas?

Dirección	Contenido
60	O
61	
62	
63	C
64	
65	
66	A
67	
68	
69	L
70	
71	
72	R
73	
74	

16. La siguiente tabla representa una pila almacenada en un bloque contiguo de celdas de memoria, tal como se explica en el texto. Si la base de la pila está en la dirección 10 y el puntero de pila contiene el valor 12, ¿qué valor nos daría una instrucción de extracción? ¿Qué valor tendrá el puntero de pila después de la operación de extracción?

Dirección	Contenido
10	F
11	C
12	A
13	B
14	E

17. Dibuje una tabla que muestre el contenido final de las celdas de memoria si la instrucción del Problema 16 hubiera sido introducir la letra D en la pila, en lugar de extraer una letra. ¿Cuál será el valor del puntero de pila después de la instrucción de inserción?

18. Diseñe un procedimiento para extraer la entrada correspondiente al fondo de una pila, de modo que se conserve el resto de la pila. Los accesos a la pila deben realizarse únicamente mediante operaciones de inser-

ción y extracción. ¿Qué estructura auxiliar de almacenamiento debe utilizarse para resolver este problema?

19. Explique el término *backtracking* con un ejemplo adecuado.

20. Suponga que disponemos de dos pilas. Si solo se nos permite mover una entrada cada vez de una pila a otra, ¿qué reordenaciones de los datos originales serían posibles? ¿Qué ordenaciones serían posibles si dispusiéramos de tres pilas?

21. Suponga que disponemos de tres pilas y que solo se nos permite mover una entrada cada vez de una pila a otra. Diseñe un algoritmo para invertir dos entradas adyacentes en una de las pilas.

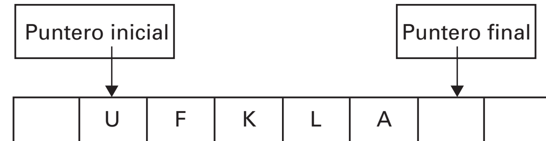
22. Suponga que queremos crear una pila de nombres, siendo la longitud de los mismos variable. ¿Por qué es conveniente almacenar los nombres en áreas separadas de la memoria y luego construir la pila con una serie de punteros a esos nombres, en lugar de permitir que la pila contenga los propios nombres?

23. ¿Cuál es la diferencia fundamental entre entre una lista enlazada y un árbol binario?

24. Suponga que deseamos implementar una "cola" en la que las nuevas entradas tengan prioridades asociadas, de modo que cada nueva entrada deberá ser colocada delante de aquellas entradas que tengan una prioridad menor. Describa un sistema de almacenamiento para implementar este tipo de "cola" y justifique las decisiones que tome.

25. Suponga que las entradas de una pila requieren una celda de memoria cada una y que el puntero de pila contiene el valor 101 en cualquier instante. ¿Cuál será el contenido del puntero de pila después de haber añadido tres entradas y eliminado dos?

26. a. Suponga que una cola implementada de forma circular se muestra en el estado indicado en el siguiente diagrama. Dibuje un diagrama que muestre la estructura después de efectuar las siguientes operaciones: insertar las letras G y R, extraer tres letras e insertar las letras D y P.



b. ¿Qué error se produciría en el apartado (a) si se insertan las letras G, R, D y P antes de elminar ninguna letra?

27. ¿Qué es una lista contigua? ¿Cuál es su limitación? ¿Cómo podría superarse esa limitación utilizando una lista enlazada?

28. Suponga que nos dan dos colas y que solo se puede desplazar cada vez una entrada desde el principio de una cola a la parte final de la otra. Diseñe un algoritmo para invertir dos entradas adyacentes en una de las colas.

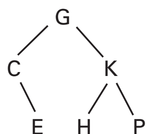
29. La siguiente tabla representa un árbol almacenado en la memoria de una máquina. Cada nodo del árbol está compuesto por tres celdas. La primera de esas celdas contiene el dato (una letra), la segunda contiene un puntero al hijo izquierdo del nodo y la tercera un puntero al hijo derecho del nodo. El puntero NIL está representado por un valor 0. Si el valor del puntero raíz es 55, dibuje una imagen del árbol.

Dirección	Contenido
40	G
41	0
42	0
43	X
44	0
45	0
46	J
47	49
48	0
49	M
50	0
51	0
52	F
53	43
54	40
55	W
56	46
57	52

30. La siguiente tabla representa el contenido de un bloque de celdas de la memoria prin-

cipal de una computadora. Observe que algunas de las celdas contienen letras del alfabeto y que cada una de esas celdas va seguida por dos celdas en blanco. Rellene las celdas en blanco de modo que el bloque de memoria represente el árbol que se muestra a continuación. Utilice la primera celda que sigue como puntero al hijo izquierdo de ese nodo y la siguiente celda como puntero al hijo derecho. Utilice el valor 0 para los punteros NIL. ¿Qué valor deberá contener el puntero raíz?

Dirección	Contenido
30	C
31	
32	
33	H
34	
35	
36	K
37	
38	
39	E
40	
41	
42	G
43	
44	
45	P
46	
47	



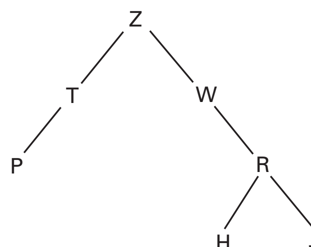
31. Explique las diferencias entre estructuras de datos estáticas y dinámicas.

32. Diseñe un algoritmo no recursivo para sustituir el algoritmo recursivo representado en la Figura 8.24. Utilice una pila para controlar cualquier paso atrás que pueda ser necesario.

33. Aplique el algoritmo recursivo de impresión de árboles de la Figura 8.24. Dibuje un diagrama que represente las activaciones anidadas del algoritmo (y la posición actual en cada una) en el momento de imprimir el nodo X.

34. Manteniendo sin variaciones el nodo raíz y sin modificar la ubicación física de los elementos de datos, cambie los punteros del árbol del Problema 29 para que el algoritmo de impresión de árboles de la Figura 8.24 imprima los nodos alfabéticamente.

35. Dibuje un diagrama que muestre cómo aparecería en la memoria el árbol binario mostrado a continuación, si se almacenara sin punteros utilizando un bloque de celdas de memoria contiguas como el descrito en la Sección 8.3.



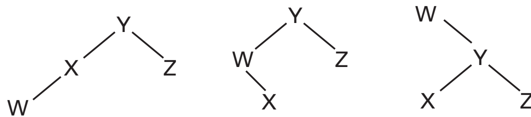
36. Suponga que las celdas contiguas que representan un árbol binario como se ha descrito en la Sección 8.3 contuvieran los valores R, B, A, D, y E, respectivamente. Dibuje un diagrama del árbol.

37. Proporcione un ejemplo en el que convendría implementar una lista (la estructura conceptual) como un árbol (la estructura real subyacente). Proporcione un ejemplo en el que convendría implementar un árbol (la estructura conceptual) en forma de lista (la estructura real subyacente).

38. Las estructuras de árbol enlazado que hemos analizado en el texto contenían punteros que permitían descender por el árbol, pasando de padre a hijo. Describa un sistema de punteros que permita ascender por el árbol, pasando de los hijos a los padres. ¿Y qué ocurre en el caso de que queramos movernos entre hermanos?

39. Describa una estructura de datos adecuada para representar una configuración durante una partida del juego de las tres en raya.

40. Indique cuáles de los árboles que se muestran a continuación se imprimirían por orden alfabético de los nodos si se aplica el algoritmo de la Figura 8.24.



41. Modifique el procedimiento de la Figura 8.24 para imprimir la "lista" en orden inverso.

42. Describa una estructura de árbol que pueda utilizarse para almacenar la historia genealógica de una familia. ¿Qué operaciones hay que realizar con el árbol? Si el árbol se implementa como una estructura enlazada, ¿qué punteros habría que asociar con cada nodo? Diseñe los procedimientos necesarios para realizar las operaciones que haya identificado, suponiendo que el árbol se implementa como una estructura enlazada con los punteros que haya decidido asociar con cada nodo. Utilizando su sistema de almacenamiento, explique cómo se podría localizar a todos los hermanos de una persona.

43. Suponga que los elementos 10, 20, 30, 40 y 50 se insertan en una cola. A continuación, se extraen tres elementos, ¿qué elementos quedarán en la cola?

44. En la implementación tradicional de un árbol, cada nodo se construye con un puntero separado para cada uno de los posibles hijos. El número de tales punteros es una decisión de diseño y representa el número máximo de hijos que cualquier nodo puede tener. Si un nodo tiene menos hijos que el número máximo de punteros admitidos, algunos de esos punteros tendrán el valor NIL. Pero ese tipo de nodo nunca puede tener más hijos que punteros. Describa cómo podría implementarse un árbol sin limitar el número de hijos que puede tener un nodo.

45. ¿Para cuáles de las siguientes operaciones debe emplearse una pila?
a. Ordenación de elementos utilizando la técnica de ordenación por burbuja.
b. Evaluación de símbolos en una expresión algebraica.
c. Pulsaciones efectuadas por el usuario de una computadora al escribir una carta.
Explique su respuesta.

46. Utilizando la instrucción de pseudocódigo `define type` presentada en la Sección 8.5, esboce una definición de un tipo abstracto de datos que represente una lista de nombres. En particular, ¿qué estructura contendrá la lista y qué procedimientos se proporcionarán para manipular la lista? (No hace falta incluir una descripción detallada de los procedimientos.)

47. Utilizando la instrucción de pseudocódigo `define type` presentada en la Sección 8.5, esboce una definición de un tipo abstracto de datos que represente una cola. A continuación, proporcione instrucciones de pseudocódigo que muestren cómo se podrían crear instancias de ese tipo y cómo podrían insertarse y borrarse entradas en dichas instancias.

48. a. ¿Cuál es la diferencia entre un tipo abstracto de datos y un tipo de datos primitivo?
b. ¿Cuál es la diferencia entre una estructura de datos de pila y una estructura de datos de cola?

49. Identifique las estructuras de datos y procedimientos que podrían aparecer en un tipo abstracto de datos que represente a un estudiante.

50. Identifique las estructuras de datos y procedimientos que podrían aparecer en un tipo abstracto de datos que represente una nave espacial simple en un videojuego.

***51.** Modifique la Figura 8.27 para que la clase defina una cola en lugar de una pila.

***52.** ¿Por qué decimos que una clase es más general que un tipo abstracto de datos tradicional?

***53.** Utilizando instrucciones de la forma DR0S y ER0S como se describe al final de la Sección 8.7, escriba una rutina completa en lenguaje máquina para insertar una entrada en una pila implementada como se muestra en la Figura 8.12. Suponga que el puntero de pila se encuentra en el registro

F y que la entrada que hay que insertar está en el registro 5.

***54.** Suponga que cada entrada de una lista enlazada está formada por una celda de memoria de datos, seguida de un puntero a la siguiente entrada de la lista. Suponga también que queremos insertar entre las entradas situadas en las posiciones B5 y C4 otra nueva entrada, situada en la dirección de memoria A0. Utilizando el lenguaje descrito en el Apéndice C y los códigos de operación adicionales D y E como se describen al final de la Sección 8.7, escriba una rutina en lenguaje máquina para llevar a cabo la inserción.

***55.** ¿Qué ventajas tiene una instrucción de la forma DR0S como se describe en la Sección 8.7 con respecto a una instrucción de la forma DRXY? ¿Qué ventaja tiene una instrucción de la forma DRXS como se describe en la Cuestión/Ejercicio 4 de la Sección 8.7 con respecto a una instrucción de la forma DR0S?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Suponga que un analista software diseña una organización de datos que permite la manipulación eficiente de los datos en una aplicación concreta. ¿Cómo pueden protegerse los derechos sobre esa estructura de datos? ¿Es una estructura de datos la expresión de una idea (como un poema) y por tanto debería estar protegida por un copyright o, por el contrario, las estructuras de datos caen en el mismo agujero legal que los algoritmos? ¿Y qué pasa con las leyes de patentes?
2. ¿Hasta qué punto puede ser peor disponer de datos incorrectos que no disponer de ningún dato?
3. En muchos programas de aplicación, el tamaño hasta el que puede crecer una pila está determinado por la cantidad de memoria disponible. Si se consume el espacio disponible, el software está diseñado para generar un mensaje como por ejemplo "desbordamiento de pila" y terminar. En la mayoría de los casos este error nunca llega a producirse y el usuario no es consciente de él. ¿Quién sería responsable si se produce ese error y, como consecuencia, se pierde información importante? ¿Cómo podría el desarrollador del software minimizar su responsabilidad legal?
4. En una estructura de datos basada en un sistema de punteros, el borrado de un elemento suele consistir en modificar el puntero en lugar de borrar celdas de memoria. Por tanto, cuando se borra una entrada de una lista enlazada, la entrada borrada sigue estando en memoria hasta que su espacio de memoria sea ocupado por otros datos. ¿Qué problemas éticos y de seguridad surgen debido a esta persistencia de los datos borrados?
5. Es fácil transferir datos y programas de una computadora a otra. Así, es sencillo transferir a muchas máquinas el conocimiento que una máquina alberga. Por el contrario, a veces hace falta mucho tiempo para que un ser humano transfiera sus conocimientos a otros. Por ejemplo, se necesita

mucho tiempo para que una persona enseñe a otra un nuevo idioma. ¿Qué implicaciones podría tener esta diferencia en cuanto a la tasa de transferencia de los conocimientos, si las capacidades de las máquinas comienzan a desafiar las capacidades de los seres humanos?

6. El uso de punteros permite enlazar datos relacionados en la memoria de una computadora de una manera que recuerda a la forma en que algunos expertos creen que la información está asociada en la mente humana. ¿En qué sentido son similares a los vínculos cerebrales esos enlaces contenidos en la memoria de una computadora? ¿En qué sentido se diferencian? ¿Es ético intentar construir computadoras que se asemejen más a la mente humana?
7. ¿Cree que la popularización de la tecnología de computadoras ha planteado nuevos problemas éticos o que simplemente ha proporcionado un nuevo contexto en el que aplicar las teorías éticas previas?
8. Suponga que el autor de un libro de texto de introducción a las Ciencias de la computación quiere incluir ejemplos de programas para ilustrar los conceptos expuestos en el texto. Sin embargo, en aras de la claridad, muchos de los ejemplos deben ser versiones simplificadas de lo que en la práctica se usaría en un software de calidad profesional. El autor sabe que los ejemplos podrían ser utilizados por un lector inocente y terminar formando parte de una aplicación software importante, en la que lo apropiado sería emplear técnicas más robustas. ¿Qué debería hacer el autor: utilizar los ejemplos simplificados, insistir en poner ejemplos muy robustos aunque eso haga disminuir su valor pedagógico o no incluir ejemplos mientras no pueda obtener a la vez tanto la claridad como la robustez?

Lecturas adicionales

Carrano, F. M. ***Data Abstraction and Problem Solving with C++: Walls and Mirrors,*** 5ª ed. Boston, MA: Addison-Wesley, 2007.

Carrano, F. M. y J. Prichard. ***Data Abstraction and Problem Solving with Java: Walls and Mirrors,*** 2ª ed. Boston, MA: Addison-Wesley, 2006.

Gray, S. ***Data Structures in Java: From Abstract Data Types to the Java Collections Framework.*** Boston, MA: Addison-Wesley, 2007.

Main, M. ***Data Structures and Other Objects Using Java,*** 3ª ed. Boston, MA: Addison-Wesley, 2006.

Main, M. y W. Savitch. ***Data Structures and Other Objects Using C++***, 4ª ed. Boston, MA: Addison-Wesley, 2010.

Shaffer, C. A. ***Practical Introduction to Data Structures and Algorithm Analysis***, 2ª ed. Upper Saddle River, NJ: Prentice Hall, 2001.

Weiss, M. A. ***Data Structures and Problem Solving Using Java***, 3ª ed. Boston, MA: Addison-Wesley, 2006.

Weiss, M. A. ***Data Structures and Algorithm Analysis in C++,*** 3ª ed. Boston, MA: Addison-Wesley, 2007.

Weiss, M. A. ***Data Structures and Algorithm Analysis in Java,*** 2ª ed. Boston, MA: Addison-Wesley, 2007.

CAPÍTULO

9

Sistemas de bases de datos

Una base de datos es un sistema que convierte un conjunto de datos de gran tamaño en una herramienta abstracta, permitiendo al usuario buscar y extraer elementos pertinentes de información, de una forma cómoda para él. En este capítulo, vamos a explorar este tema, así como hacer incursiones en los campos de la minería de datos, que buscan técnicas para descubrir patrones ocultos en grandes conjuntos de datos, y de las estructuras de archivos tradicionales, que proporcionan muchas de las herramientas que se utilizan en los sistemas actuales de bases de datos y de minería de datos.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

La tecnología actual es capaz de almacenar cantidades enormemente grandes de datos, pero esas colecciones son inútiles a menos que seamos capaces de extraer aquellos elementos concretos de información que sean pertinentes para la tarea que tenemos entre manos. En este capítulo vamos a estudiar los sistemas de bases de datos y a aprender cómo esos sistemas aplican la abstracción con el fin de convertir grandes conglomerados de datos en fuentes de información útil. Como tema relacionado, investigaremos el campo de la minería de datos, que se está expandiendo rápidamente y cuyo objetivo consiste en desarrollar técnicas para identificar y explorar patrones dentro de conjuntos de datos. También examinaremos los principios en que se basan las estructuras de archivos tradicionales, que proporcionan el fundamento para los sistemas actuales de bases de datos y de minería de datos.

9.1 Fundamentos de las bases de datos

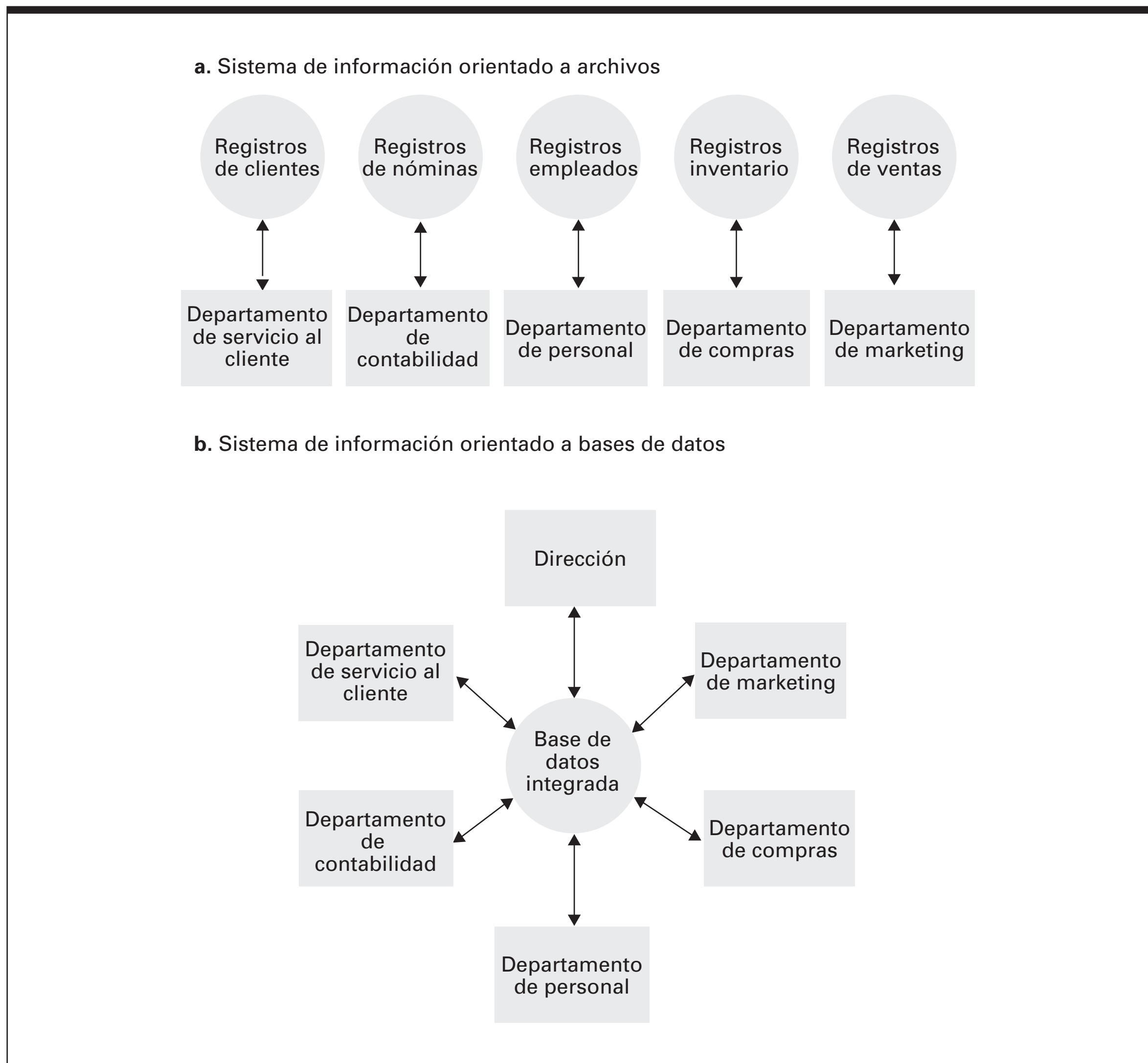
El término **base de datos** hace referencia a un conjunto de datos multidimensional en el sentido de que los enlaces internos existentes entre los distintos elementos hacen que se pueda acceder a la información con diversas perspectivas. Esto difiere de un sistema de archivos tradicional (Sección 9.5), que en ocasiones se denomina **archivo plano**, que no es más que un sistema de almacenamiento unidimensional, lo que quiere decir que presenta la información que contiene desde un único punto de vista. Mientras que un archivo plano que contenga información acerca de compositores de música y sus obras podría proporcionar una lista de obras musicales ordenadas por compositor, una base de datos podría presentar todas las obras de un mismo compositor, todos los compositores que hayan escrito algún tipo concreto de música y quizá también los compositores que hayan escrito variaciones de las obras de algún otro compositor.

La importancia de los sistemas de bases de datos

Históricamente, a medida que las computadoras fueron encontrando usos cada vez más amplios en el campo de la gestión de información, cada aplicación tendía a ser implementada como un sistema separado con su propio conjunto de datos. Las nóminas se procesaban utilizando el archivo de nóminas, el departamento de personal mantenía sus propios registros de empleados y el inventario se gestionaba mediante un archivo de inventario. Esto significaba que buena parte de la información requerida por una organización estaba duplicada por toda la empresa y que muchos elementos diferentes, pero relacionados, se almacenaban en sistemas separados. En este tipo de entorno, los sistemas de bases de datos surgieron como medio de integrar la información almacenada y mantenida por una organización concreta (Figura 9.1). Con este tipo de sistema, los mismos datos de ventas podían utilizarse para generar órdenes de compra de materias primas, generar informes sobre las tendencias del mercado, dirigir los anuncios de productos a los clientes que más probablemente pudieran responder de forma favorable a esa información y generar los bonos de los miembros del equipo comercial.

Estos conjuntos integrados de información proporcionaban un valioso recurso con el que tomar decisiones de gestión, suponiendo que pudiera acce-

Figura 9.1 Comparación entre una organización en archivos y la organización mediante bases de datos.



derse a la información de una manera significativa. A su vez, la investigación en el campo de las bases de datos se centró en el desarrollo de técnicas que permitieran aportar la información contenida en una base de datos al proceso de toma de decisiones. Se han hecho muchos progresos a este respecto. Hoy día, la tecnología de bases de datos, combinada con las técnicas de minería de datos constituye una importante herramienta de gestión, permitiendo a la dirección de una organización extraer información relevante a partir de enormes cantidades de datos que cubren todos los aspectos relativos a la organización y su entorno.

Además, los sistemas de bases de datos se han convertido en la tecnología subyacente que da soporte a muchos de los sitios más populares de la World Wide Web. El objetivo fundamental de sitios como Google, eBay y Amazon es

proporcionar una interfaz entre los clientes y las bases de datos. Para responder a la solicitud de un cliente, el servidor interroga a una base de datos, organiza los resultados en forma de página web y envía dicha página al cliente. Dichas interfaces web han popularizado un nuevo papel para la tecnología de bases de datos, en el que una base de datos ya no es un medio de almacenar los registros de una empresa, sino que se convierte en el propio producto de la empresa. De hecho, al combinar la tecnología de bases de datos con las interfaces web, Internet se ha convertido en una de las principales fuentes de información mundiales.

El papel de los esquemas

Entre las desventajas de la proliferación de tecnologías de bases de datos se encuentra la posibilidad de que personal no autorizado acceda a datos confidenciales. Alguien que está realizando un pedido en el sitio web de una empresa no debe tener acceso a los datos financieros de la empresa. De forma similar, un empleado del departamento de personal de la empresa puede necesitar acceder a los registros de empleados, pero no debería acceder a los registros de inventario o de ventas. Por tanto, la capacidad de controlar el acceso a la información contenida en la base de datos es tan importante como la capacidad de compartir esa información.

Para proporcionar a los diferentes usuarios acceso a diferentes tipos de informaciones dentro de una base de datos, los sistemas de bases de datos suelen utilizar lo que se denomina esquemas y subesquemas. Un **esquema** es una descripción de toda la estructura de la base de datos; el software de la base de datos utiliza esa descripción para mantener la base de datos. Un **subesquema** es una descripción de aquella parte de la base de datos que es relevante para las necesidades de un usuario concreto. Por ejemplo, un esquema para una base de datos de una universidad indicaría que cada registro de estudiante contiene elementos tales como la dirección y el teléfono actuales de dicho estudiante, además de su historial académico. Asimismo, indicaría que el registro de cada estudiante está enlazado con el registro de su tutor. A su vez, el registro de cada profesor de la universidad contendría la dirección de esa persona, el historial laboral, etc. Basándose en este esquema, se mantendría un sistema de enlaces que permitiera en último término conectar la información relativa a un estudiante con el historial laboral de un profesor.

Para impedir que el administrativo encargado de las matrículas de los estudiantes de la universidad utilice este sistema de enlaces con el fin de obtener información privilegiada acerca de los profesores, el acceso de ese administrativo a la base de datos debe restringirse a un subesquema cuya descripción de los registros de los profesores no incluya el historial laboral. Con este subesquema, el administrativo podría averiguar qué profesor es el tutor de un estudiante concreto, pero no podría obtener acceso a información adicional acerca del profesor. Por el contrario, el subesquema correspondiente al departamento de contabilidad proporcionaría el historial laboral de cada profesor, pero no incluiría la relación entre estudiantes y tutores. Por tanto, el departamento de contabilidad podría modificar el salario de un profesor pero no podría obtener los nombres de los estudiantes de los que es tutor.

Sistemas de gestión de bases de datos

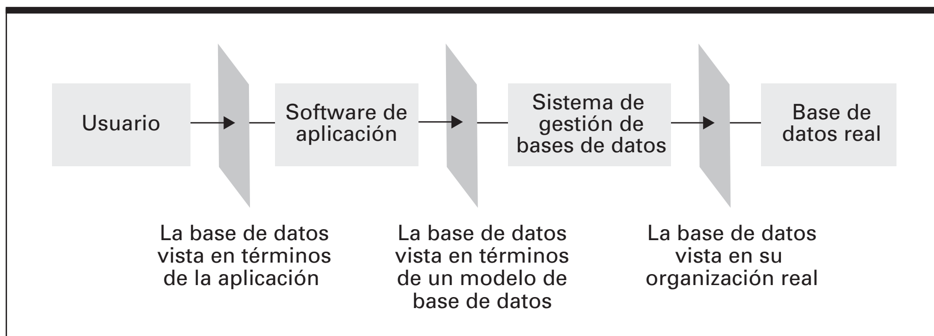
Una aplicación de bases de datos típica consta de varias capas de software, que agruparemos en dos capas principales: una capa de aplicación y una capa de gestión de base de datos (Figura 9.2). El software de aplicación gestiona la comunicación con el usuario de la base de datos y puede ser bastante complejo, como ilustran las aplicaciones en las que los usuarios acceden a una base de datos por medio de un sitio web. En tal caso, la capa completa de aplicación está compuesta por clientes dispersos por Internet y por un servidor que utiliza la base de datos con el fin de satisfacer las solicitudes de los clientes.

Observe que el software de aplicación no manipula directamente la base de datos. La manipulación real de la misma se lleva a cabo mediante el **sistema de gestión de base de datos** (*DBMS, Database Management System*). Una vez que el software de aplicación ha determinado qué acción está solicitando el usuario, utiliza el DBMS como herramienta abstracta para obtener el resultado. Si la solicitud consiste en añadir o borrar datos, será el DBMS el que modifique en la práctica la base de datos. Si la solicitud consiste en extraer información, será el DBMS el que lleve a cabo las búsquedas requeridas.

Esta dicotomía entre el software de aplicación y el DBMS tiene varias ventajas. Una de ellas es que permite la construcción y uso de herramientas abstractas que, como ya hemos visto, constituyen uno de los principales conceptos simplificadores en el campo del diseño de software. Si se aíslan dentro del DBMS los detalles relativos a cómo está realmente almacenada la base de datos, el diseño del software de aplicación puede simplificarse enormemente. Por ejemplo, con un DBMS bien diseñado, el software de aplicación no tiene que preocuparse de si la base de datos está almacenada en una única máquina o está dispersa entre muchas máquinas conectadas en red, como **base de datos distribuida**. En lugar de ello, será el DBMS el que trate con estos problemas, permitiendo que el software de aplicación acceda a la base de datos sin preocuparse por dónde están realmente almacenados los datos.

Un segunda ventaja de separar el software de aplicación del DBMS es que este tipo de organización proporciona un medio de controlar el acceso a la base de datos. Obligando a que sea el DBMS el que realice todos los accesos a la base de datos, puede imponer las restricciones definidas por los distintos subesquemas. En particular, el DBMS puede utilizar el esquema completo de la base

Figura 9.2 Las capas conceptuales de una implementación de base de datos.



Bases de datos distribuidas

Con los avances relativos a las capacidades de red, los sistemas de bases de datos han crecido hasta abarcar una serie de bases de datos que se conocen con el nombre de bases de datos distribuidas y que consisten en datos que residen en diversas máquinas simultáneamente. Por ejemplo, una corporación internacional podría almacenar y mantener los registros de los empleados locales en los sitios locales, pero enlazar dichos registros por medio de una red, con el fin de crear una única base de datos distribuida.

Una base de datos distribuida puede contener datos fragmentados y/o replicados. El primer caso está ilustrado por el ejemplo anterior relativo a los registros de empleados, en el que diferentes fragmentos de la base de datos se almacenaban en diferentes aplicaciones. En el segundo caso, se almacenan en diferentes ubicaciones una serie de duplicados del mismo componente de base de datos. Esa replicación puede realizarse con el fin de reducir el tiempo necesario para extraer la información. Ambos casos plantean problemas que no están presentes en las bases de datos centralizadas más tradicionales: por ejemplo, cómo ocultar la naturaleza distribuida de la base de datos para que funcione como un sistema coherente o cómo garantizar que las partes replicadas de una base de datos continúen siendo copias exactas unas de otras a medida que se producen las actualizaciones. Además, el estudio de las bases de distribuidas es una de las áreas de investigación más activas en la actualidad.

de datos para sus necesidades internas, pero exigir que el software de aplicación empleado por cada usuario permanezca dentro de los límites descritos por el subesquema correspondiente a ese usuario.

Otra razón más para separar la interfaz de usuario y las tareas reales de manipulación de los datos en dos capas diferentes de software es la de conseguir la denominada **independencia de los datos**, la capacidad de modificar la organización de la propia base de datos sin tener que modificar el software de aplicación. Por ejemplo, el departamento de personal puede tener que agregar un campo al registro de cada empleado para indicar si ese empleado ha decidido participar en el nuevo programa de cobertura sanitaria de la empresa. Si el software de aplicación tratara directamente con la base de datos, ese cambio en el formato de los datos podría requerir que se hicieran modificaciones en todos los programas de aplicación que traten con la base de datos. Como resultado, la modificación impuesta por el departamento de personal podría provocar cambios tanto en el programa de nóminas como por ejemplo en el programa de impresión de etiquetas de correo para el boletín mensual de la empresa.

Esa separación entre el software de aplicación y un DBMS elimina la necesidad de modificar los programas. Para implementar un cambio en la base de datos requerido por un único usuario, lo único que hace falta es modificar el esquema global y los subesquemas de aquellos usuarios que se vean afectados por el cambio. Los subesquemas de los restantes usuarios seguirán siendo iguales, así que su software de aplicación, que está basado en esos subesquemas no modificados, no tendrá que ser alterado.

Modelos de bases de datos

Ya hemos visto repetidamente cómo se puede utilizar la abstracción para ocultar la complejidad interna de un componente. Los sistemas de gestión de bases de datos proporcionan un ejemplo más de este concepto. Estos sistemas ocultan la complejidad de la estructura interna de una base de datos, permitiendo al usuario de la misma imaginarse que la información almacenada en la base de datos está organizada con un formato más útil. En particular, un DBMS contiene rutinas que traducen los comandos expresados en términos de una vista conceptual de la base de datos a las acciones requeridas por el sistema real de almacenamiento de los datos. Esta vista conceptual de la base de datos se conoce como **modelo de base de datos**.

En las siguientes secciones consideraremos tanto el modelo de base datos relacional como el modelo de base de datos orientado a objetos. En el caso del modelo de base de datos relacional, la vista conceptual de la base de datos es la de una colección de tablas compuestas de filas y columnas. Por ejemplo, la información acerca de los empleados de una empresa puede visualizarse como una tabla que contiene una fila por empleado y una serie de columnas etiquetadas como nombre, dirección, número de identificación del empleado, etc. A su vez, el DBMS dispondrá de rutinas que permitirán al software de aplicación seleccionar ciertas entradas de una fila concreta de la tabla o quizá informar del rango de valores existente en la columna de salario (aún cuando la información no esté en realidad almacenada en filas y columnas).

Estas rutinas forman las herramientas abstractas utilizadas por el software de aplicación a la hora de acceder a la base de datos. Para ser más precisos, el software de aplicación suele escribirse en uno de los lenguajes de programación de propósito general existentes, como los presentados en el Capítulo 6. Estos lenguajes proporcionan los ingredientes básicos para las expresiones algorítmicas, pero carecen de instrucciones para manipular una base de datos. Sin embargo, un programa escrito en uno de esos lenguajes puede utilizar las rutinas proporcionadas por el DBMS en forma de subrutinas pre-escritas, lo que tiene el efecto en la práctica de ampliar las capacidades del lenguaje de forma tal que permite dar soporte a la imagen conceptual del modelo de base de datos.

La búsqueda de mejores modelos de bases de datos continúa siendo un campo de investigación activo. El objetivo es encontrar modelos que permitan conceptualizar fácilmente sistemas de datos complejos, que conduzcan a formas concisas de expresar las solicitudes de información y que produzcan sistemas de gestión de bases de datos eficientes.

Cuestiones y ejercicios

1. Identifique dos departamentos de una planta de fabricación que pudieran tener diferentes usos para la misma información de inventario o para una información similar. A continuación, describa cómo podría diferir el subesquema para los dos departamentos.
2. ¿Cuál es el propósito de un modelo de base de datos?
3. Resuma los papeles del software de aplicación y de un DBMS.

9.2 El modelo relacional

En esta sección vamos a examinar más de cerca del modelo de base de datos relacional. Este modelo representa los datos como si estuvieran almacenados en tablas rectangulares, denominadas **relaciones**, que son similares al formato en el que se visualiza la información en los programas de hoja de cálculo. Por ejemplo, el modelo relacional permite representar la información relativa a los empleados de una empresa mediante una relación como la mostrada en la Figura 9.3.

Una fila de una relación se denomina **tupla**. En la relación de la Figura 9.3, las tuplas están compuestas por la información acerca de un empleado concreto. Las columnas de una relación se denominan **atributos** porque cada entrada de una columna describe alguna característica, o atributo, de la entidad representada por la correspondiente tupla.

Problemas del diseño relacional

Un paso fundamental a la hora de diseñar una base de datos relacional es diseñar las relaciones que forman la base de datos. Aunque esto puede parecer una tarea simple, hay muchos aspectos sutiles esperando a atrapar a los diseñadores incautos.

Suponga que además de la información contenida en la relación de la Figura 9.3, queremos incluir información acerca de los puestos de trabajo que ocupan los empleados. Podríamos desear incluir un historial laboral asociado con cada empleado que constara de atributos tales como el puesto ocupado (secretaria, director de oficina, supervisor de planta), un código de identificación del puesto (que diferencia unos puestos de otros), el código de la categoría asociado a cada puesto, el departamento al que pertenece ese puesto y el periodo durante el que el empleado ha ocupado el puesto en términos de una fecha de inicio y una fecha de terminación. (Utilizaremos un asterisco como fecha de terminación si se trata del puesto actual del empleado.)

Una solución a este problema consistiría en ampliar la relación de la Figura 9.3 para incluir estos atributos como columnas adicionales de la tabla, como se muestra en la Figura 9.4. Sin embargo, un examen más atento del resultado permite detectar varios problemas. El primero es una falta de eficiencia debido a la redundancia. La relación ya no contiene solo una tupla por cada empleado, sino una tupla por cada asignación de un puesto de trabajo o de un empleado. Si un empleado ha ido progresando en la empresa a través de una secuencia de varios puestos de trabajo, varias tuplas de la nueva relación deberán contener la

Figura 9.3 Una relación que contiene información sobre los empleados.

Id Empl	Nombre	Direccion	NSS
25X15	Juan Barrio	San Juan 33	111223333
34Y70	Charo Garrido	Av Principal 563	999009999
23Y34	Gervasio Salas	Dr. Circo 1555	111005555
•	•	•	•
•	•	•	•
•	•	•	•

misma información acerca del empleado (nombre, dirección, número de identificación y número de la Seguridad Social). Por ejemplo, la información personal acerca de los empleados Barrio y Salas está repetida porque han ocupado más de un puesto. Además, cuando un puesto concreto ha sido ocupado por numerosos empleados, es preciso repetir en cada tupla que represente una asignación de ese puesto de trabajo tanto la información correspondiente al departamento asociado con ese puesto como el código de categoría apropiado. Por ejemplo, la descripción del puesto de director de planta está duplicada porque hay más de un empleado que ha ocupado este puesto.

Otro problema quizá más serio de esta relación ampliada es el que surge cuando consideramos el problema de borrar información de la base de datos. Por ejemplo, suponga que Juan Barrio es el único empleado que tiene el puesto identificado como D7. Si abandonara la empresa y lo borráramos de la base de datos representada en la Figura 9.4, perderíamos la información relativa al puesto de trabajo D7. Después de todo, la única tupla que contiene el hecho de que el puesto D7 requiere una categoría K2 es la tupla relacionada con Juan Barrio.

Podría argumentarse que la capacidad de borrar solamente una parte de una tupla resolvería el problema, pero esto introduciría otras complicaciones. Por ejemplo, ¿deberíamos también conservar la información relativa al puesto de trabajo F5 en una tupla parcial, o esta información ya se encuentra en algún otro lugar dentro de la relación? Además, la tentación de usar tuplas parciales es un sólido indicio de que puede mejorarse el diseño de la base de datos.

La causa de todos estos problemas es que hemos combinado más de un concepto en una misma relación. Tal como está propuesta, la relación ampliada de la Figura 9.4 contiene información que trata directamente con los empleados (nombre, número de identificación, dirección, número de la Seguridad Social), información acerca de los puestos disponibles en la empresa (identificación del puesto, título del puesto, departamento y código de categoría) e información relativa a la relación entre los empleados y los puestos de trabajo (fecha de inicio y fecha de terminación). Teniendo en cuenta esta

Figura 9.4 Una relación que contiene redundancia.

ID Empl	Nombre	Direccion	NSS	Id Puesto	Tit Puesto	Cod Cat	Dept	Fecha Inicio	Fecha Fin
25X15	Juan Barrio	San Juan 33	111223333	F5	Director planta	FM3	Ventas	1-9-2009	30-9-2010
25X15	Juan Barrio	San Juan 33	111223333	D7	Resp. dept.	K2	Ventas	1-10-2010	*
34Y70	Charo Garrido	Av. Principal 563	999009999	F5	Director planta	FM3	Ventas	1-10-2009	*
23Y34	Gervasio Salas	Dr. Circo 1555	111005555	S25X	Secretario	T5	Personal	1-3-1999	30-4-2010
23Y34	Gervasio Salas	Dr. Circo 1555	111005555	S26Z	Secretario	T6	Contabilidad	1-5-2010	*
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Sistemas de bases de datos para PC

Las computadoras personales se utilizan en una amplia variedad de aplicaciones, que van desde las más elementales a las muy sofisticadas. En las aplicaciones de "bases de datos" elementales, como por ejemplo almacenar la lista de felicitaciones de Navidad o mantener los registros de la liga de fútbol del colegio, se utilizan a menudo sistemas de hoja de cálculo en lugar de software de base de datos, ya que la aplicación requiere poco más que la capacidad de almacenar, imprimir y ordenar los datos. Sin embargo, existen verdaderos sistemas de bases de datos para el mercado del PC, siendo uno de ellos Microsoft Access. Es un sistema de base de datos relacional completo, tal como se describe en la Sección 9.2, junto con un software de generación de diagramas e informes. Access proporciona un ejemplo excelente de cómo los principios presentados en este texto forman la base de productos muy populares en el mercado actual.

observación podemos resolver nuestros problemas rediseñando el sistema con tres relaciones: una para cada una de las categorías anteriores. Podemos mantener la relación original de la Figura 9.3 (que ahora denominaremos relación `EMPLEADO`) e insertar la información adicional mediante dos nuevas relaciones denominadas `PUESTO` y `ASIGNACION`, lo que nos da la base de datos de la Figura 9.5.

Una base de datos compuesta por estas tres relaciones contiene la información pertinente acerca de los empleados en la relación `EMPLEADO`, la información sobre los puestos disponibles en la relación `PUESTO` y sobre el historial laboral en la relación `ASIGNACION`. La información adicional está disponible implícitamente combinando la información procedente de distintas relaciones. Por ejemplo, si conocemos el número de identificación de un empleado, podemos conocer los departamentos en que ese empleado ha trabajado, localizando primero todos los puestos que el empleado ha ocupado mediante la relación `ASIGNACION` y localizando después los departamentos asociados con esos puestos, por medio de la relación `PUESTO` (Figura 9.6). Mediante procesos como estos, cualquier información que pudiera obtenerse a partir de esa única relación de gran tamaño podrá ahora obtenerse a partir de las tres relaciones más pequeñas y sin los problemas citados.

Lamentablemente, dividir la información en varias relaciones no siempre es tan sencillo como en el ejemplo anterior. Por ejemplo, compare la relación original de la Figura 9.7 con los atributos `IdEmpl`, `TitPuesto` y `Dept`, con la descomposición propuesta en dos relaciones. A primera vista, el sistema de dos relaciones parece contener la misma información que el sistema de una sola relación, pero de hecho no es así. Considere por ejemplo el problema de localizar el departamento en el que trabaja un cierto empleado. Esto puede hacerse fácilmente en el sistema de una sola relación, consultando la tupla que contiene el número de identificación del empleado deseado y extrayendo el departamento correspondiente. Sin embargo, en el sistema de dos relaciones, la información deseada no está disponible necesariamente. Podemos encontrar el título del puesto del empleado deseado y un departamento que tenga dicho puesto, pero esto no implica necesariamente que el empleado buscado trabaje en ese departamento concreto, porque podría haber varios departamentos con puestos que tuvieran el mismo título.

Figura 9.5 Una base de datos de empleados compuesta por tres relaciones.

relación EMPLEADO

Id Empl	Nombre	Direccion	NSS
25X15	Juan Barrio	San Juan 33	111223333
34Y70	Charo Garrido	Av. Principal 563	999009999
23Y34	Gervasio Salas	Dr. Circo 1555	111005555

relación PUESTO

Id Puesto	Tit Puesto	Cod Cat	Dept
S25X	Secretario	T5	Personal
S26Z	Secretario	T6	Contabilidad
F5	Director planta	FM3	Ventas
•	•	•	•
•	•	•	•
•	•	•	•

relación ASIGNACION

Id Empl	Id Puesto	Fecha Inicio	Fecha Fin
23Y34	S25X	1-3-1999	30-4-2010
34Y70	F5	1-10-2009	*
23Y34	S26Z	1-5-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

Vemos, por tanto, que en ocasiones dividir una relación en relaciones más pequeñas provoca la pérdida de información, mientras que en otras ocasiones no es así (este último caso se denomina **descomposición sin pérdidas**). Estas características relacionales son muy importantes a la hora de acometer un diseño. El objetivo es identificar las características relacionales que pueden conducir a problemas en el diseño de bases de datos y encontrar formas de reorganizar dichas relaciones para eliminar esas características problemáticas.

Operaciones relacionales

Ahora que tenemos una comprensión básica de cómo pueden organizarse los datos en términos del modelo relacional, es el momento de ver cómo puede extraerse información de una base de datos compuesta por relaciones. Comenzaremos examinando algunas operaciones que podríamos querer llevar a cabo con relaciones.

En ocasiones, necesitamos seleccionar ciertas tuplas de una relación. Para extraer la información acerca de un empleado, tendremos que seleccionar la tupla de la relación `EMPLEADO`, que tenga el valor apropiado en el atributo correspondiente del número de identificación del empleado. O bien, para obtener una lista de los puestos de trabajo de un cierto departamento, tendremos

Figura 9.6 Localización de los departamentos en los que el empleado 23Y34 ha trabajado.

relación EMPLEADO

Id Empl	Nombre	Direccion	NSS
25X15	Juan Barrio	San Juan 33	111223333
34Y70	Charo Garrido	Av. Principal 563	999009999
23Y34	Gervasio Salas	Dr. Circo 1555	111005555
•	•	•	•
•	•	•	•
•	•	•	•

relación PUESTO

Id Puesto	Tit Puesto	Cod Cat	Dept
S25X	Secretario	T5	Personal
S26Z	Secretario	T6	Contabilidad
F5	Director planta	FM3	Ventas
•	•	•	•
•	•	•	•
•	•	•	•

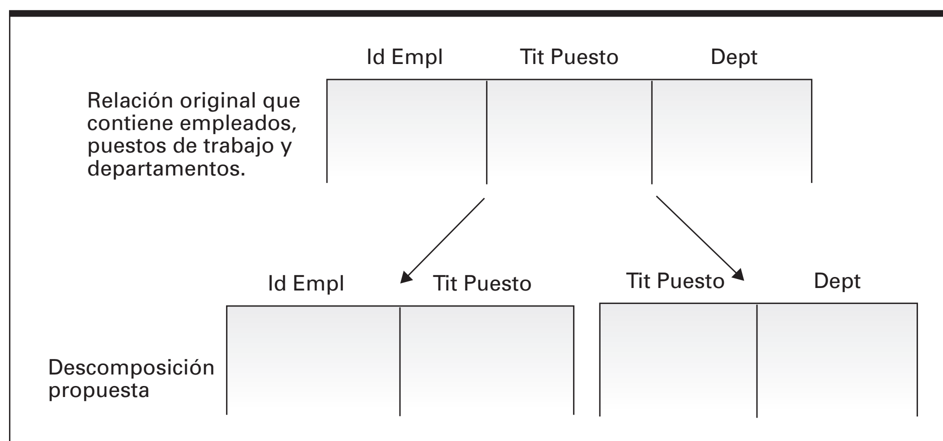
corresponden a los departamentos de personal y contabilidad.

relación ASIGNACION

Id Empl	Id Puesto	Fecha Inicio	Fecha Fin
23Y34	S25X	3-1-1999	4-30-2010
34Y70	F5	10-1-2009	*
23Y34	S26Z	5-1-2010	*
•	•	•	•
•	•	•	•
•	•	•	•

Los puestos ocupados por el empleado 23Y34

Figura 9.7 Una relación y una descomposición propuesta.



que seleccionar las tuplas de la relación PUESTO que tengan dicho departamento como atributo de departamento. El resultado de este proceso es otra relación compuesta por las tuplas seleccionadas en la relación padre. El resul-

tado de seleccionar información acerca de un empleado concreto nos dará una relación que contendrá una única tupla extraída de la relación EMPLEADO. El resultado de seleccionar las tuplas asociadas con un cierto departamento nos dará como resultado una relación que probablemente contendrá varias tuplas de la relación PUESTO.

En resumen, una operación que podemos desear llevar a cabo con una relación consiste en seleccionar tuplas que posean ciertas características y colocar dichas tuplas seleccionadas en una nueva relación. Para expresar esta operación, adoptaremos la sintaxis

```
NUEVA ← SELECT from EMPLEADO where IdEmpl = "34Y70"
```

La semántica de esta sentencia consiste en crear una nueva relación denominada NUEVA que contenga las tuplas (solo debería haber una en este caso) de la relación EMPLEADO cuyo atributo IdEmpl sea igual a 34Y70 (Figura 9.8).

A diferencia de la operación SELECT, que extrae filas de una relación, la operación PROJECT extrae columnas. Por ejemplo, suponga que al buscar los títulos de puestos de un cierto departamento, ya hemos seleccionado las tuplas de la relación PUESTO correspondientes al departamento objetivo y que hemos colocado esas tuplas en una relación denominada NUEVA1. La lista que estaremos buscando será la columna TitPuesto dentro de esta nueva relación. La operación PROJECT nos permite extraer esta columna (o varias columnas si hiciera falta) y colocar el resultado en una nueva relación. Podemos expresar ese tipo de operación mediante

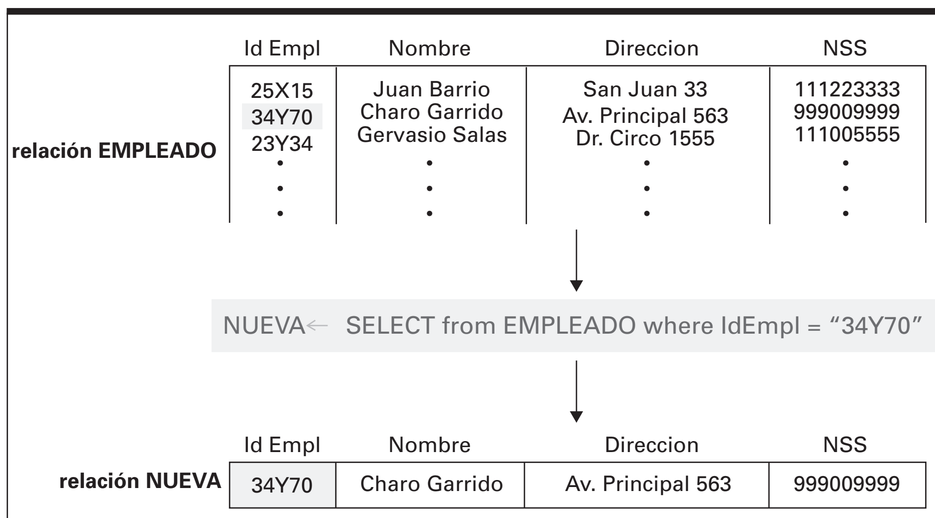
```
NUEVA2 ← PROJECT TitPuesto from NUEVA1
```

El resultado es la creación de una nueva relación (denominada NUEVA2) que contiene una única columna de valores, correspondientes a la columna TitPuesto de la relación NUEVA1.

Veamos otro ejemplo de la operación PROJECT. La sentencia

```
CORREO ← PROJECT Nombre, Direccion from EMPLEADO
```

Figura 9.8 La operación SELECT.



puede utilizarse para obtener un listado de los nombres y direcciones de todos los empleados. Esta lista se almacenará en la nueva relación (de dos columnas) denominada CORREO (Figura 9.9).

Otra operación que se utiliza en conjunción con las bases de datos relacionales es la operación JOIN. Se usa para combinar diferentes relaciones en una única relación. El resultado de combinar mediante JOIN dos relaciones produce una nueva relación cuyos atributos son los atributos de las relaciones originales (Figura 9.10). Los nombres de esos atributos serán los mismos que los de las relaciones originales, salvo porque cada uno tendrá como prefijo la relación original de la que procede. (Si la relación A que tiene los atributos V y W se combina mediante JOIN con la relación B, que contiene los atributos X, Y y Z, entonces el resultado tendrá cinco atributos denominados A.V, A.W, B.X, B.Y y B.Z.) Este convenio de denominación garantiza que los atributos de la nueva relación tenga nombres distintivos, incluso aunque las relaciones originales pudieran tener nombres de atributos en común.

Las tuplas (filas) de la nueva relación se generan concatenando las tuplas de las dos relaciones originales (véase de nuevo la Figura 9.10). Para determinar qué tuplas se combinan realmente, con el fin de formar tuplas de la nueva relación, se añade una condición para la aplicación de JOIN. Una de las posibles condiciones es que determinados atributos designados tengan el mismo valor. De hecho, este es el caso representado en la Figura 9.10, en el que se ilustra el resultado de ejecutar la sentencia

```
C ← JOIN A and B where A.W = B.X
```

En este ejemplo habrá que concatenar una tupla de la relación A con una tupla de la relación B únicamente en aquellos casos en los que los atributos W y X de las dos tuplas sean iguales. Por tanto, la concatenación de la tupla (r, 2) de la

Figura 9.9 La operación PROJECT.

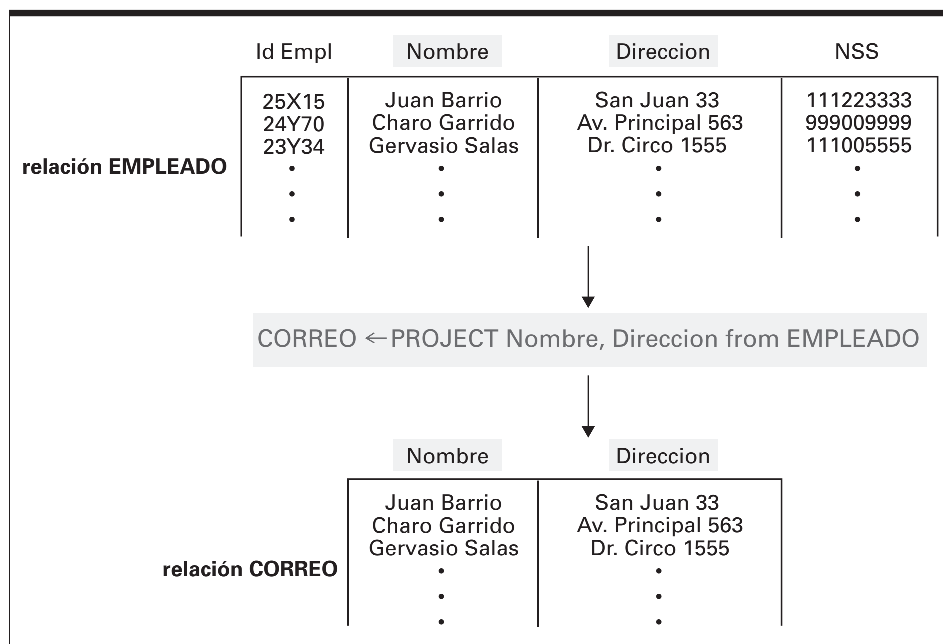
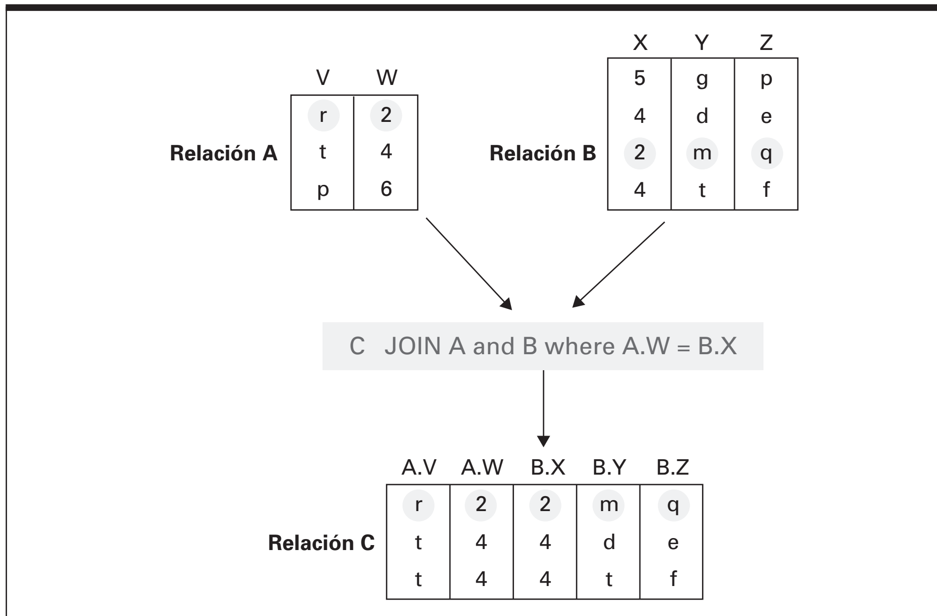


Figura 9.10 La operación JOIN.



relación A con la tupla (2, m, q) de la relación B aparece en el resultado, porque el valor del atributo W de la primera es igual al valor del atributo X en la segunda. Por el contrario, el resultado de concatenar la tupla (r, 2) de la relación A con la tupla (5, g, p) de la relación B no aparece en la relación final, porque estas tuplas no comparten valores comunes en los atributos W y X.

Veamos otro ejemplo: la Figura 9.11 representa el resultado de ejecutar la sentencia

```
C ← JOIN A and B where A.W < B.X
```

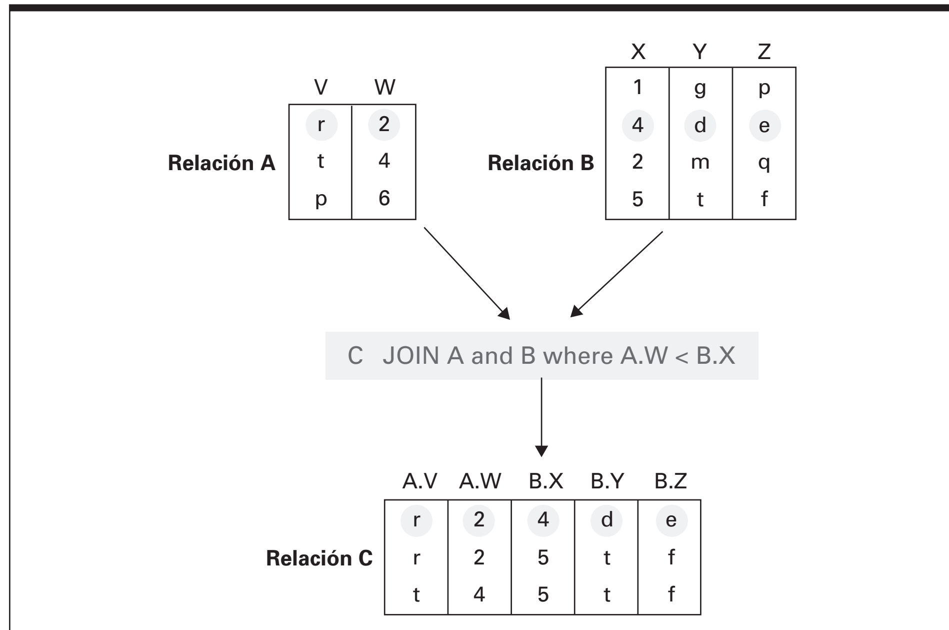
Observe que las tuplas del resultado son exactamente aquellas en las que el atributo W de la relación A es menor que el atributo X de la relación B.

Veamos ahora cómo puede utilizarse la operación JOIN con la base de datos de la Figura 9.5 para obtener un listado de todos los números de identificación de los empleados junto con el departamento en el que trabaja cada empleado. La primera observación que haremos es que la información requerida está distribuida entre más de una relación, por lo que el proceso de extraer la información requerirá algo más que selecciones y proyecciones. De hecho, la herramienta que necesitamos es la sentencia

```
NUEVA1 ← JOIN ASIGNACION and PUESTO
            where ASIGNACION.IdPuesto = PUESTO.IdPuesto
```

que genera la relación NUEVA1, como se muestra en la Figura 9.12. A partir de esta relación, nuestro problema puede resolverse seleccionando primero aquellas tuplas en las que ASIGNACION.FechaFin sea igual a "*" (lo que indica que es el puesto actual del empleado) y luego proyectando los atributos ASIGNACION.IdEmpl y PUESTO.Dept. En resumen, la información que necesitamos puede obtenerse a partir de la base de datos de la Figura 9.5 ejecutando la secuencia

Figura 9.11 Otro ejemplo de la operación JOIN.



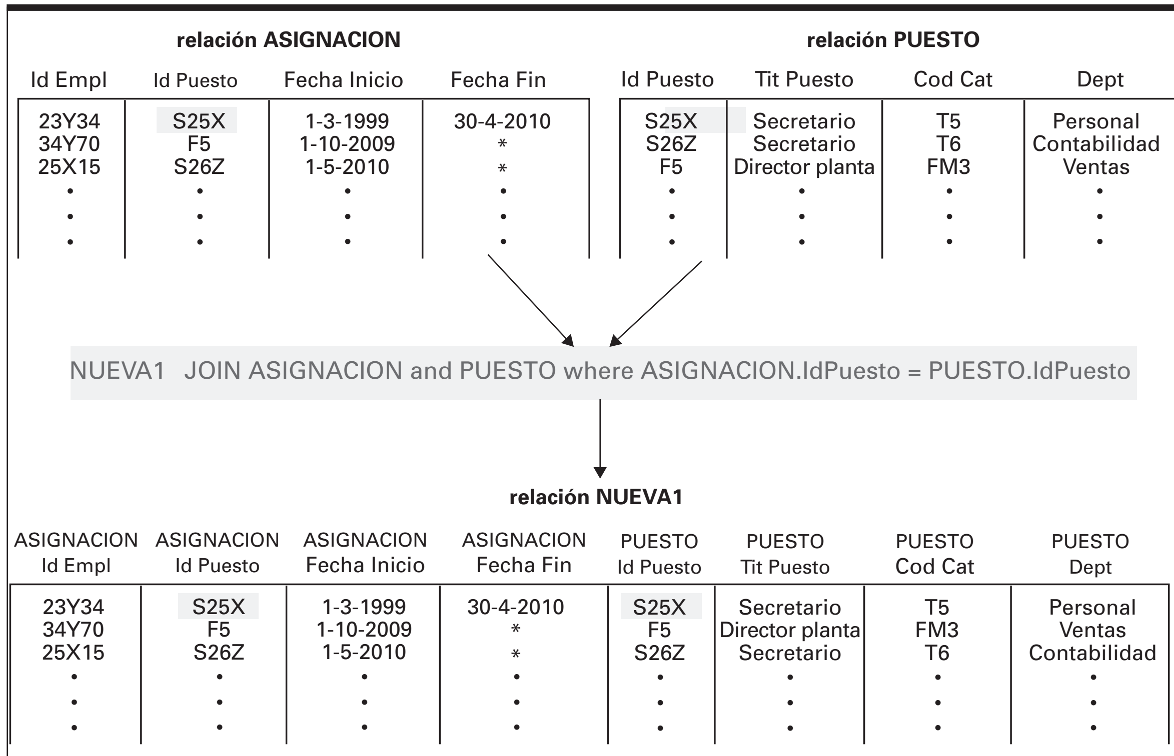

```
NUEVA1 ← JOIN ASIGNACION and PUESTO
         where ASIGNACION.IdPuesto = PUESTO.IdPuesto
NUEVA2 ← SELECT from NUEVA1 where ASIGNACION.FechaFin = "*"
LIST ← PROJECT ASIGNACION.IdEmpl, PUESTO.Dept from NUEVA2
```

SQL

Ahora que hemos presentado las operaciones relacionales básicas, reconsideremos la estructura global de un sistema de base de datos. Recuerde que una base de datos se almacena en realidad en un sistema de almacenamiento masivo. Para evitar que el programador de aplicaciones tenga que conocer los detalles de dichos sistemas, se proporciona un sistema de gestión de bases de datos que permite escribir el software de aplicación en términos de un modelo de base de datos, tal como el modelo relacional. El DBMS acepta comandos en términos del modelo elegido y los convierte en acciones relativas a la estructura de almacenamiento real. Esta conversión se gestiona mediante un conjunto de rutinas dentro del DBMS, que son utilizadas como herramientas abstractas por el software de aplicación. Así, un DBMS basado en el modelo relacional incluiría rutinas para realizar las operaciones SELECT, PROJECT y JOIN, que podrían ser invocadas desde el software de aplicación. De esta forma, el software de aplicación podría escribirse como si la base de datos estuviera realmente almacenada en la forma tabular simple del modelo relacional.

Los sistemas de gestión de bases de datos relacionales actuales no necesariamente proporcionan rutinas para realizar las operaciones SELECT, PROJECT y JOIN en su forma básica. En su lugar, proporcionan rutinas que pueden ser combinaciones de estos pasos básicos. Un ejemplo es el lenguaje SQL (***Structured Query Language***, Lenguaje de consulta estructurado), que forma la base

Figura 9.12 Una aplicación de la operación JOIN.



de la mayoría de los sistemas de consulta de bases de datos relacionales. Por ejemplo, SQL es el lenguaje subyacente en el sistema de bases de datos relacional MySQL utilizado por muchos servidores de bases de datos en Internet.

Una de las razones de la popularidad de SQL es que el Instituto ANSI lo ha estandarizado. Otra razón es que originalmente fue desarrollado y comercializado por IBM y se ha aprovechado así de muchos años de experiencia de uso. En esta sección explicamos cómo se expresan en SQL las consultas a una base de datos relacional.

Aunque vamos a ver que una consulta en SQL se expresa en una forma más bien imperativa, la realidad es que es esencialmente una sentencia declarativa. Trate de leer las sentencias SQL como descripciones de la información deseada más que como secuencias de las actividades que hay que realizar. La importancia de este aspecto es que SQL ahorra a los programadores de aplicaciones el engorro de desarrollar algoritmos para manipular relaciones, los programadores simplemente tienen que describir la información deseada.

Como primer ejemplo de sentencia SQL, reconsideremos nuestra última consulta en la que hemos desarrollado un proceso en tres pasos para obtener todos los números de identificación de los empleados junto con sus correspondientes departamentos. En SQL, esta consulta completa podría representarse mediante la única sentencia

```
select IdEmpl, Dept
from ASIGNACION, PUESTO
```

```
where ASIGNACION.IdPuesto = PUESTO.IdPuesto
  and ASIGNACION.FechaFin = '*'
```

Como indica este ejemplo, cada sentencia de consulta SQL puede contener tres cláusulas: una cláusula `select`, una cláusula `from` y una cláusula `where`. En términos generales, lo que dicha sentencia está haciendo es solicitar el resultado de combinar mediante JOIN todas las relaciones enumeradas en la cláusula `from`, seleccionar mediante SELECT aquellas tuplas que satisfagan las condiciones indicadas en la cláusula `where` y luego proyectar mediante PROJECT aquellas tuplas enumeradas en la cláusula `select`. (Observe que la terminología está en cierto modo invertida, ya que la cláusula `select` de una sentencia SQL identifica los atributos utilizados en la operación PROJECT.) Veamos algunos ejemplos sencillos.

La sentencia

```
select Nombre, Direccion
from EMPLEADO
```

genera un listado de todos los nombres y direcciones de los empleados contenidos en la relación EMPLEADO. Observe que esto es simplemente una operación PROJECT.

La sentencia

```
select EmplId, Nombre, Direccion, NumSSN
from EMPLEADO
where Nombre = 'Charo Garrido'
```

genera toda la información de la tupla asociada con Charo Garrido en la relación EMPLEADO. Esto es esencialmente una operación SELECT.

La sentencia

```
select Nombre, Direccion
from EMPLEADO
where Nombre = 'Charo Garrido'
```

genera el nombre y la dirección de Charo Garrido contenidos en la relación EMPLEADO. Esto es una combinación de las operaciones SELECT y PROJECT.

La sentencia

```
select EMPLEADO.Nombre, ASIGNACION.FechaInicio
from EMPLEADO, ASIGNACION
where EMPLEADO.EmplId = ASIGNACION.IdEmpl
```

genera un listado de todos los nombres de empleado y las fechas en las que comenzaron a trabajar en ese puesto. Observe que esto es el resultado de combinar mediante JOIN las relaciones EMPLEADO y ASIGNACION y luego seleccionar mediante SELECT y PROJECT las tuplas y atributos apropiados, identificados en las cláusulas `where` y `select`.

Cerramos esta sección resaltando que SQL dispone de sentencias para definir la estructura de las relaciones, crear relaciones y modificar el contenido de las relaciones, además de para realizar consultas. A continuación proporcionamos algunos ejemplos de las sentencias `insert into`, `delete from` y `update`.

La sentencia

```
insert into EMPLEADO
values ('42Z12', 'Susana Burgos', 'San Pedro 33',
        '444661111')
```

añade una tupla a la relación EMPLEADO que contiene los valores indicados;

```
delete from EMPLEADO
where Name = 'Gervasio Salas'
```

borra de la relación EMPLEADO la tupla relativa a Gervasio Salas y

```
update EMPLEADO
set Direccion = 'Av. Secundaria 1812.'
where Nombre = 'Juan Barrio'
```

modifica la dirección de la tupla asociada con Juan Barrio en la relación EMPLEADO.

Cuestiones y ejercicios

1. Responda a las siguientes cuestiones basándose en la información parcial proporcionada en las relaciones EMPLEADO, PUESTO y ASIGNACION de la Figura 9.5:
   a. ¿Quién es el secretario del departamento de contabilidad que tiene experiencia en el departamento de personal?
   b. ¿Quién es el director de planta en el departamento de ventas?
   c. ¿Qué puesto ocupa actualmente Gervasio Salas?
2. Basándose en las relaciones EMPLEADO, PUESTO y ASIGNACION presentadas en la Figura 9.5, escriba una secuencia de operaciones relacionales para obtener una lista de todos los títulos de los puestos de trabajo dentro del departamento de personal.
3. Basándose en las relaciones EMPLEADO, PUESTO y ASIGNACION presentadas en la Figura 9.5, escriba una secuencia de operaciones relacionales para obtener una lista de los nombres de todos los empleados junto con los departamentos correspondientes.
4. Convierta a SQL sus respuestas a las Cuestiones 2 y 3.
5. ¿En qué sentido proporciona independencia de los datos el modelo relacional?
6. ¿Cómo se enlazan entre sí las diferentes relaciones de una base de datos relacional?

9.3 Bases de datos orientadas a objetos

Otro modelo de base de datos es el basado en el paradigma de orientación a objetos. Esta solución conduce a obtener una **base de datos orientada a objetos**, compuesta por objetos que están enlazados entre sí con el fin de reflejar sus relaciones. Por ejemplo, una implementación orientada a objetos de la base de datos de empleados de la sección anterior podría estar compuesta por tres clases (tipos de objetos): EMPLEADO, PUESTO y ASIGNACION. Un objeto de la clase

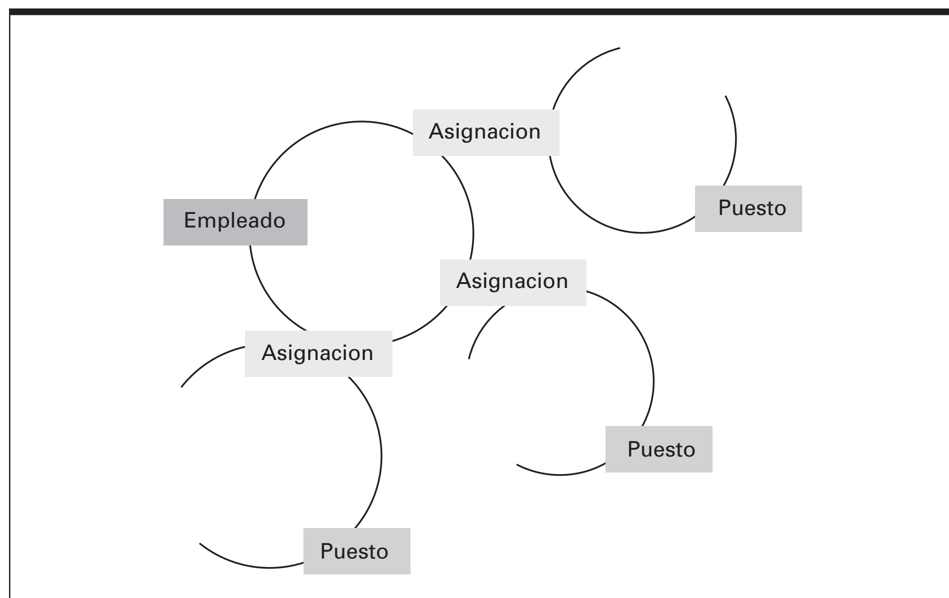
EMPLEADO contendría entradas tales como IdEmpl, Nombre, Direccion y NSS; un objeto de la clase PUESTO contendría entradas tales como IdPuesto, TitPuesto, CodCat y Dept; y cada objeto de la clase ASIGNACION podría contener entradas como FechaInicio y FechaFin.

En la Figura 9.13 se muestra una representación conceptual de ese tipo de base datos, en la que los enlaces entre los distintos objetos están representados mediante líneas que conectan los objetos relacionados. Si nos centramos en un objeto de tipo EMPLEADO, lo vemos enlazado a un conjunto de objetos de tipo ASIGNACION, que representa las diversas asignaciones de puesto de trabajo que ese empleado en concreto ha tenido. A su vez, cada uno de estos objetos de tipo ASIGNACION está enlazado a un objeto de tipo PUESTO, que representa el puesto asociado con dicha asignación. Así, todas las asignaciones de un empleado pueden encontrarse siguiendo los enlaces que van desde el objeto que representa a ese empleado. De forma similar, pueden encontrarse todos los empleados que han ocupado un puesto concreto siguiendo los enlaces correspondientes al objeto que representa ese puesto de trabajo.

Los enlaces entre objetos en una base de datos orientada a objetos suelen ser mantenidos por el DBMS, por lo que el programador no tiene que preocuparse por los detalles de cómo se implementan estos enlaces a la hora de escribir el software de aplicación. En lugar de ello, cuando se añade un nuevo objeto a la base de datos, el software de aplicación simplemente especifica los otros objetos a los que debe estar enlazado. El DBMS crea entonces cualquier sistema de enlaces que pueda ser necesario con el fin de registrar esas asociaciones. En particular, un DBMS puede enlazar los objetos que representan las asignaciones de un determinado empleado de forma similar a una lista enlazada.

Otra tarea de un DBMS orientado a objetos es proporcionar almacenamiento permanente para los objetos que se le confían, un requisito que puede parecer obvio, pero que es inherentemente distinto de la forma en que normal-

Figura 9.13 Las asociaciones entre objetos en una base de datos orientada a objetos.



mente se tratan los objetos. Por lo general, cuando se ejecuta un programa orientado a objetos, los objetos creados durante la ejecución del programa se descartan una vez que este termina. En este sentido, los objetos se consideran transitorios, pero los objetos creados y añadidos a una base de datos deben guardarse después de que haya terminado el programa que los creó. Tales objetos se denominan **persistentes**. Por ello, la creación de objetos persistentes se aparta significativamente de la norma.

Los defensores de las bases de datos orientadas a objetos ofrecen numerosos argumentos para demostrar por qué la solución orientada a objetos es mejor en el diseño de bases de datos que el enfoque tradicional. Uno de esos argumentos es que la solución orientada a objetos permite que todo el sistema software (software de aplicación, DBMS y la propia base de datos) se diseñe bajo el mismo paradigma. Esto contrasta con la práctica tradicional de utilizar un lenguaje de programación imperativo para desarrollar el software de aplicación con el fin de interrogar a una base de datos relacional. Inherente a dicha tarea es el conflicto entre los paradigmas imperativo y relacional. Esta distinción es quizá demasiado sutil para el nivel con el que estamos abordando nuestro estudio, pero esa diferencia ha provocado muchos errores de software a lo largo de los años. Incluso a nuestro nivel, no es difícil apreciar que una base de datos orientada a objetos combinada con un programa de aplicación orientado a objetos, permite obtener una imagen homogénea de objetos que se comunican unos con otros a través del sistema. Por el contrario, una base de datos relacional combinada con un programa de aplicación imperativo sugiere la imagen de dos organizaciones inherentemente distintas tratando de encontrar una interfaz común.

Para entender otra ventaja que las bases de datos orientadas a objetos tienen con respecto a sus equivalentes relacionales, considere el problema de almacenar los nombres de los empleados en una base de datos relacional. Si almacenamos un nombre completo en forma de un único atributo en una relación, entonces las consultas relativas únicamente a los apellidos serán muy complicadas. Sin embargo, si almacenamos el nombre en forma de atributos individuales, tal como nombre de pila, primer apellido y segundo apellido, entonces el número de atributos se hace problemático, porque no todos los nombres se adaptan a una estructura específica, aún cuando la población esté restringida a una única cultura. En una base de datos orientada a objetos, estos problemas pueden ocultarse dentro del objeto que almacene el nombre del empleado. El nombre del empleado podrá guardarse como un objeto inteligente que será capaz de informar de cuál es el nombre del empleado relacionado en diversos formatos. De ese modo, desde fuera de esos objetos, sería igual de fácil tratar solo con los apellidos que con los nombres completos, los nombres de pila o los alias. Los detalles relacionados con cada una de las distintas perspectivas estarían encapsulados dentro de los objetos.

Esta capacidad de encapsular los aspectos técnicos de los diferentes formatos de datos resulta ventajosa también en otros casos. En una base de datos relacional, los atributos de una relación forman parte del diseño global de la base de datos, por lo que los tipos asociados con esos atributos influyen sobre toda la base de datos. (Las variables para el almacenamiento temporal deben declararse con el tipo apropiado y es preciso designar procedimientos para manipular datos de los diversos tipos.) Así, ampliar una base de datos relacional para incluir atributos de nuevos tipos (audio y vídeo) puede ser problemático. En

particular, puede que sea necesario ampliar diversos procedimientos dispersos por todo el diseño de la base de datos, con el fin de incorporar estos nuevos tipos de datos. Sin embargo, en un diseño orientado a objetos, los mismos procedimientos utilizados para extraer un objeto que represente el nombre de un empleado pueden utilizarse para extraer un objeto que represente una película, porque las distinciones de tipo pueden ocultarse dentro de los objetos implicados. Por tanto, la solución orientada a objetos es más compatible con la construcción de bases de datos multimedia, una característica que ya está demostrando sus grandes ventajas.

Otra ventaja más que el paradigma orientado a objetos ofrece para el diseño de bases de datos es el potencial para almacenar objetos inteligentes, en lugar de simplemente datos. Es decir, un objeto puede contener métodos que describan cómo debería responder el objeto a mensajes relativos a su contenido y a sus relaciones. Por ejemplo, cada objeto de la clase `EMPLEADO` de la Figura 9.13 podría contener métodos para leer y actualizar la información contenida en el objeto, así como un método para leer el historial laboral del empleado y quizá un método para modificar la asignación de puesto de trabajo de ese empleado. De la misma forma, cada objeto de la clase `PUESTO` podría tener un método para leer los datos específicos del puesto de trabajo y quizá otro método para informar de qué empleados han ocupado ese puesto de trabajo en concreto. Así, para extraer el historial laboral de un empleado, no necesitaríamos construir un procedimiento elaborado. En lugar de ello, podríamos simplemente pedir al objeto de empleado apropiado que informara acerca de su historial laboral. Esta capacidad de construir bases de datos cuyos componentes respondan de forma inteligente a las consultas ofrece múltiples y excitantes posibilidades que van mucho más alla de las disponibles con las bases de datos relacionales tradicionales.

Cuestiones y ejercicios

1. ¿Qué métodos contendría una instancia de un objeto de la clase `ASIGNACION` en la base de datos de empleados que hemos descrito en esta sección?
2. ¿Qué es un objeto persistente?
3. Identifique algunas clases, junto con algunas de sus características internas, que puedan emplearse en una base de datos orientada a objetos utilizada para gestionar el inventario de un almacén.
4. Identifique una ventaja que una base de datos orientada a objetos pueda tener si la comparamos con una base de datos relacional.

9.4 Mantenimiento de la integridad de una base de datos

Los sistemas baratos de gestión de bases de datos para uso personal son sistemas relativamente simples. Tienden a tener un único objetivo (ocultar al usuario los detalles técnicos de la implementación de la base de datos). Las bases de

datos mantenidas por esos sistemas son relativamente pequeñas y por lo general contienen información cuya pérdida o corrupción podría ser incómoda, pero raras veces desastrosa. Cuando surge de verdad un problema, el usuario puede normalmente corregir los errores directamente o recargar la base de datos a partir de una copia de seguridad para luego efectuar manualmente las correcciones requeridas con el fin de actualizar dicha copia. Este proceso puede ser incómodo, pero el coste de evitar esa incomodidad tiende a ser mayor que la incomodidad en sí. En cualquier caso, esa incomodidad está restringida a solo unas pocas personas y las pérdidas financieras asociadas suelen ser limitadas.

Sin embargo, en el caso de los sistemas de bases de datos comerciales, multiusuario y de gran tamaño, los riesgos son mucho mayores. El coste de tener datos incorrectos o de que se pierdan los datos puede ser enorme y tener consecuencias devastadoras. En estos entornos, uno de los principales papeles del DBMS es mantener la integridad de la base de datos protegiéndola frente a problemas tales como operaciones que por alguna razón solo lleguen a completarse parcialmente o como la posibilidad de que diversas operaciones interfieran de manera inadvertida, haciendo que al final la base de datos contenga información incorrecta. Este es el papel del DBMS que vamos a analizar en esta sección.

El protocolo de confirmación/anulación (*commit/rollback*)

Una única transacción, como por ejemplo una transferencia de fondos de una cuenta bancaria a otra, la cancelación de una reserva en una línea aérea o la matrícula de un estudiante en un curso universitario, puede implicar múltiples pasos en el nivel de la base de datos. Por ejemplo, una transferencia de fondos entre cuentas bancarias requiere que se reduzca el saldo de una de las cuentas y se incremente correspondientemente el saldo de la otra. En el tiempo que media entre esos pasos, la información de la base de datos puede ser incoherente. De hecho, el dinero habrá desaparecido durante el breve periodo que media entre el momento de disminuir el saldo de la primera cuenta y el momento de aumentar el saldo de la segunda. De la misma forma, cuando se reasigna el asiento de un pasajero en un vuelo, puede haber un instante en el que el pasajero no tenga asiento asignado o un instante en el que la lista de pasajeros parezca tener un pasajero más de lo que realmente tiene.

En el caso de bases de datos grandes sujetas a una gran carga de transacciones es bastante probable que cualquier instantánea tomada en un momento aleatorio encuentre a la base de datos en mitad de la realización de alguna transacción. Por tanto, cualquier solicitud para que se ejecute una transacción o cualquier error de funcionamiento de un equipo es probable que se produzca en algún instante en el que la base de datos se encuentra en un estado incoherente.

Consideremos primero el problema de un error de funcionamiento. El objetivo del DBMS es garantizar que ese tipo de problemas no dejen a la base de datos en un estado incoherente permanente. Esto suele llevarse a cabo manteniendo un registro en un sistema de almacenamiento no volátil, como por ejemplo un disco magnético, en el que se van anotando todas las actividades de las transacciones. Antes de permitir a una transacción modificar la base de

datos, se anota en el registro la modificación que se va a realizar. De ese modo, el registro contendrá un archivo permanente de todas las acciones de cada transacción.

El punto en el que todos los pasos que componen una transacción se han anotado en el registro se denomina **punto de confirmación**. Es en ese momento cuando el DBMS dispone de la información que necesita para reconstruir la transacción por su propia cuenta en caso de que llegara a ser necesario. En ese punto, el DBMS "se compromete" con la transacción en el sentido de que acepta la responsabilidad de garantizar que las actividades de la transacción se reflejarán en la base de datos. En el caso de que se produzca un fallo en un equipo, el DBMS puede utilizar la información contenida en su registro para reconstruir las transacciones que se han completado (confirmado) desde que se llevó a cabo la última copia de seguridad.

Si surgen problemas antes de que una transacción haya alcanzado su punto de confirmación, el DBMS puede encontrarse con una transacción parcialmente ejecutada que no puede completarse. En este caso, el registro puede utilizarse para **anular** (deshacer) las actividades realizadas por la transacción. En el caso de un error de funcionamiento, el DBMS podría recuperarse anulando aquellas transacciones que estuvieran incompletas (no confirmadas) en el momento en que se produjo el error.

La anulación de transacciones no está restringida, sin embargo, al proceso de recuperarse de errores de funcionamiento de los equipos. De hecho, suelen ser parte de la operación normal de un DBMS. Por ejemplo, una transacción podría terminarse antes de haber completado sus pasos debido a un intento de acceder a información privilegiada. O puede que se vea implicada en un interbloqueo, en el que una serie de transacciones competidoras se encuentran esperando por los datos que están usando las otras. En estos casos, el DBMS puede utilizar el registro para anular una transacción y evitar así errores de base de datos debido a la existencia de transacciones incompletas.

Para enfatizar la delicada naturaleza del diseño de sistemas DBMS, es necesario recalcar que existen sutiles problemas inherentes al proceso de anulación. La anulación de una transacción puede afectar a las entradas de la base de datos que hayan sido utilizadas por otras transacciones. Por ejemplo, la transacción que se está anulando podría haber actualizado un saldo de una cuenta bancaria y otra transacción podría haber ya basado sus actividades en ese valor actualizado. Esto podría significar que esas transacciones adicionales tengan que ser también anuladas, lo que puede a su vez afectar a otras transacciones. El resultado es el problema conocido como de la **anulación en cascada.**

Bloqueo (*locking*)

Consideremos ahora el problema de la ejecución de una transacción mientras que la base de datos se encuentra en un estado intermedio de otra transacción, una situación que puede llevar a interacciones inadvertidas entre las transacciones y producir resultados erróneos. Por ejemplo, puede surgir el problema conocido con el nombre de **problema de totalización incorrecta** si una transacción se encuentra en mitad del proceso de transferir dinero de una cuenta bancaria a otra al mismo tiempo que otra transacción trata de calcular el total

de los depósitos existentes en el banco. Esto podría resultar en un total que sea demasiado grande o demasiado pequeño, dependiendo del orden en que se efectúen los pasos de la transferencia. Otra posibilidad es la que se conoce con el nombre del **problema de la actualización perdida**, que podemos ejemplificar mediante dos transacciones, cada una de las cuales efectúa un pago contra una misma cuenta. Si una de las transacciones lee el saldo actual de la cuenta justo en el instante en el que la otra acaba de leer el saldo pero no ha calculado todavía el nuevo, entonces ambas transacciones basarán su cálculo del nuevo saldo en el mismo valor de saldo inicial. Por ello, el efecto de uno de los pagos no se verá reflejado en la base de datos.

Para resolver estos problemas, un DBMS podría obligar a las transacciones a que se ejecuten en su totalidad una a una, manteniendo a todas las nuevas transacciones en una cola hasta que las que la preceden hayan completado su tarea. Pero una transacción invierte a menudo una gran cantidad de tiempo esperando a que se realicen operaciones con el almacenamiento masivo. Entremezclando la ejecución de las transacciones, el tiempo durante el que una transacción está esperando puede ser utilizado por otra transacción para procesar datos que haya extraído. Por ello, la mayoría de los grandes sistemas de gestión de bases de datos disponen de un planificador para coordinar la compartición de tiempo entre transacciones, de forma bastante similar a como un sistema operativo de multiprogramación coordina la intercalación de procesos (véase la Sección 3.3).

Para protegerse frente anomalías tales como el problema de la totalización incorrecta y el problema de la actualización perdida, esos planificadores incorporan un **protocolo de bloqueo** mediante el que los elementos de una base de datos que estén siendo actualmente utilizados por alguna transacción se marcan como en uso. Estas marcas se denominan bloqueos y decimos que los elementos marcados están bloqueados. Existen dos tipos comunes de bloqueos: **bloqueos compartidos** y **bloqueos exclusivos**. Estos bloqueos se corresponden con los dos tipos de acceso a los datos que puede necesitar una transacción (acceso compartido y acceso exclusivo). Si una transacción no va a modificar un elemento de datos, entonces necesitará un acceso compartido, lo que quiere decir que otras transacciones también estarán autorizadas a ver los datos. Sin embargo, si la transacción va a modificar el elemento, deberá tener acceso exclusivo, lo que quiere decir que debe ser la única transacción con acceso a esos datos.

En un protocolo de bloqueo, cada vez que una transacción solicita acceso a un elemento de datos, también debe informar al DBMS del tipo de acceso que requiere. Si una transacción solicita acceso compartido a un elemento que no está bloqueado o que está bloqueado con un acceso compartido, entonces el DBMS concede el acceso y el elemento se marca con un bloqueo compartido. Sin embargo, si el elemento solicitado ya está marcado con un acceso exclusivo, el acceso adicional se deniega. Si una transacción solicita acceso exclusivo a un elemento, esa solicitud solo se concede si el elemento no tiene ningún bloqueo asociado. De esta manera, una transacción que vaya a alterar los datos protegerá a esos datos frente a otras transacciones obteniendo acceso exclusivo, mientras que múltiples transacciones pueden compartir el acceso a un elemento si ninguna de ellas va a modificarlo. Por supuesto, una vez que una

transacción ha terminado con un elemento, se lo notifica al DBMS y este elimina el bloqueo asociado.

Para gestionar el caso en el que se rechaza la solicitud de acceso efectuada por una transacción hay disponibles varios métodos. Uno de ellos es que la transacción se ve forzada simplemente a esperar hasta que el elemento solicitado pasa a estar disponible. Sin embargo, esta solución puede conducir a un interbloqueo, ya que dos transacciones que requieran acceso exclusivo a los mismos dos elementos de datos podrían bloquearse entre sí, si cada una de ellas obtiene acceso exclusivo a uno de los elementos y luego insiste en esperar a obtener acceso al otro. Para evitar esos interbloqueos, algunos sistemas de gestión de bases de datos dan prioridad a las transacciones más antiguas. Es decir, si una transacción más antigua solicita acceso a un elemento que está bloqueado por una transacción más reciente, se obliga a esta última a liberar todos sus elemento de datos y sus actividades se deshacen, basándose en el registro. Después, a la transacción antigua se le proporciona acceso al elemento que solicitó y se fuerza a la transacción más reciente a comenzar de nuevo. Si a causa de este mecanismo, una transacción reciente es forzada a anularse repetidamente, se irá haciendo más antigua durante el proceso y terminará siendo una de las transacciones con mayor prioridad. Este protocolo, conocido con el nombre de **protocolo de desalojo y espera** (*wound-wait*, las transacciones más antiguas desalojan a las más recientes y las más recientes esperan a que las más antiguas terminen), garantiza que todas las transacciones tengan la oportunidad de terminar completando su tarea.

Cuestiones y ejercicios

1. ¿Cuál es la diferencia entre una transacción que ha alcanzado su punto de confirmación y otra que no lo ha hecho?
2. ¿Cómo podría un DBMS protegerse frente a un proceso de anulación en cascada masivo?
3. Muestre cómo el entremezclado incontrolado de dos transacciones, una que tiene que deducir 100 euros de una cuenta y la otra que tiene deducir 200 euros de la misma cuenta, podría producir saldos finales de 100, 200 y 300 euros, suponiendo que el saldo inicial sea de 400 euros.
4. **a.** Resuma los posibles resultados de una transacción que solicite acceso compartido a un elemento de una base de datos.
   b. Resuma los posibles resultados de una transacción que solicite acceso exclusivo a un elemento de una base de datos.
5. Describa la secuencia de sucesos que conduciría a un interbloqueo entre transacciones que estén realizando operaciones en un sistema de base de datos.
6. Describa cómo podría romperse el interbloqueo de su respuesta a la Cuestión 5. ¿Requeriría su solución utilizar el registro del sistema de gestión de la base de datos? Razone su respuesta.

9.5 Estructuras de archivo tradicionales

En esta sección vamos a dejar momentáneamente de lado el estudio de los sistemas de base de datos multidimensionales, con el fin de considerar las estructuras de archivo tradicionales. Estas estructuras representan el inicio histórico de los sistemas de almacenamiento y recuperación de datos a partir de los cuales ha evolucionado la tecnología actual de base de datos. Muchas de las técnicas desarrolladas para estas estructuras (como las de indexación y *hashing*) son herramientas importantes en la construcción de las bases de datos masivas y complejas, que se utilizan en la actualidad.

Archivos secuenciales

Un **archivo secuencial** es un archivo al que se accede de forma serie, desde su principio hasta su final, como si la información del archivo estuviera dispuesta en una única fila de gran longitud. Entre los ejemplos podemos citar los archivos de audio, los archivos de vídeo, archivos que contienen programas y archivos que contienen documentos de texto. De hecho, la mayoría de los archivos creados por un usuario típico de una computadora personal son archivos secuenciales. Por ejemplo, cuando se guarda una hoja de cálculo, su información se codifica y almacena como un archivo secuencial, a partir del cual el software de manejo de hojas de cálculo puede reconstruir la hoja de cálculo guardada.

Los archivos de texto, que son archivos secuenciales en los que cada registro lógico es un único símbolo codificado mediante ASCII o Unicode, suelen servir como herramienta básica para construir archivos secuenciales más elaborados, como los archivos de registros de empleados. Lo único que hace falta es establecer un formato uniforme para la representación de la información relativa a cada empleado en forma de cadena de texto; codificar esa información con dicho formato y luego grabar los registros de empleados resultantes uno detrás de otro, como una única cadena de texto. Por ejemplo, podríamos construir un archivo de empleados simple decidiendo introducir cada registro de empleado como una cadena de 31 caracteres, formada por un campo de 25 caracteres que contendrá el nombre del empleado (relleno con los caracteres de espacio suficientes como para completar el campo de 25 caracteres), seguido por un campo de 6 caracteres que representará el número de identificación de ese empleado. El archivo final sería una larga cadena de caracteres codificados en la que cada bloque de 31 caracteres representaría la información relativa a un único empleado (Figura 9.14). La información se extraería del archivo en términos de los registros lógicos, compuestos por bloques de 31 caracteres. Dentro de cada uno de estos bloques, los campos individuales se identificarían de acuerdo con el formato uniforme empleado para construir los bloques.

Los datos de un archivo secuencial deben guardarse en el almacenamiento masivo de forma tal que la naturaleza secuencial del archivo se conserve. Si el sistema de almacenamiento masivo es secuencial (como sucede, por ejemplo, con una cinta magnética o un CD), esta tarea será muy sencilla. Simplemente tendremos que guardar el archivo en el medio de almacenamiento de acuerdo con las propiedades secuenciales de dicho medio. Después, para procesar el archivo, simplemente habrá que leer y procesar los datos contenidos en el

Figura 9.14 La estructura de un archivo de empleado simple implementado como un archivo de texto.

mismo, en el orden en que se encuentran. Este es exactamente el proceso que se sigue a la hora de reproducir discos CD de audio, en los que la música se almacena como un archivo secuencial sector por sector, a lo largo de una pista continua en espiral.

Sin embargo, en el caso de un sistema de almacenamiento en disco magnético, el archivo estará disperso por diferentes sectores que podrían ser leídos en órdenes distintos. Para preservar el orden correcto, la mayoría de los sistemas operativos (o para ser más precisos, el administrador de archivos) mantienen una lista de los sectores en los que está almacenado el archivo. Esta lista se guarda como parte del sistema de directorios del disco en el mismo disco que el archivo. Por medio de esta lista, el sistema operativo puede extraer los sectores en la secuencia apropiada como si el archivo estuviera almacenado secuencialmente, aunque en realidad el archivo esté distribuido por varias partes del disco.

Inherente al procesamiento de un archivo secuencial está la necesidad de detectar cuándo se ha alcanzado el final del archivo. Genéricamente, nos referimos al final de un archivo secuencial mediante el término **fin de archivo** (EOF, *end-of-file*). Existen varias formas de identificar el EOF. Una de ellas consiste en insertar un registro especial, denominado **centinela**, al final del archivo. Otra forma consiste en utilizar la información del sistema de directorios del sistema operativo para identificar el final de un archivo. Es decir, como el sistema operativo sabe qué sectores contiene el archivo, también sabe dónde termina el archivo.

Un ejemplo clásico relativo a la utilización de archivos secuenciales es el procesamiento de las nóminas en una pequeña empresa. Vamos a imaginarnos de cara a este ejemplo un archivo secuencial compuesto por una serie de registros lógicos, cada uno de los cuales contiene la información acerca del salario de un empleado (nombre, número de identificación, escala salarial, etc.), a partir de la cual deben imprimirse los cheques correspondientes de forma periódica.

A medida que se extrae cada registro de empleado, se calcula la nómina de ese empleado y se genera el cheque correspondiente. La actividad de procesar ese tipo de archivo secuencial se podría ejemplificar mediante la sentencia

```
while (EOF no se ha alcanzado) do
  (extraer el siguiente registro del archivo y procesarlo)
```

Cuando los registros lógicos dentro de un archivo secuencial se identifican mediante valores clave de los campos, el archivo suele disponerse de modo que los registros aparezcan en el orden determinado por esos valores clave (que quizá sea alfabético o numérico). Ese tipo de disposición simplifica la tarea de procesamiento de la información del archivo. Por ejemplo, suponga que procesar las nóminas requiere que se actualice el registro de cada empleado para reflejar la información contenida en el archivo de asignaciones horarias de ese empleado. Si tanto el archivo que contiene los datos horarios como el que contiene los registros de empleados están en el mismo orden de acuerdo con las mismas claves, entonces este proceso de actualización puede gestionarse accediendo a ambos archivos secuencialmente, utilizando el registro horario extraído de un archivo para actualizar el registro correspondiente del otro archivo. Esta es una mejora significativa con respecto al proceso repetido de búsqueda que se necesitaría si los archivos no estuvieran en el mismo orden. Debido a ello, la actualización de archivos secuenciales clásicos suele realizarse en múltiples pasos. En primer lugar, se registra la nueva información (como por ejemplo el conjunto de asignaciones horarias) en un archivo secuencial que se conoce como archivo de transacción y este archivo de transacción se ordena para hacerlo corresponder con el orden del archivo que hay que actualizar, que se denomina archivo maestro. Después, se actualizan los registros del archivo maestro extrayendo secuencialmente los registros de ambos archivos.

Una ligera variante de este proceso de actualización es el proceso de combinar dos archivos secuenciales para formar un nuevo archivo que contenga los registros de los dos archivos originales. Suponemos que los registros de los archivos de entrada están ordenados de manera ascendente, de acuerdo con un campo clave común y también asumimos que hay que combinar los archivos de forma que se genere un archivo de salida cuyas claves también estén en

Figura 9.15 Un procedimiento para combinar dos archivos secuenciales.

```
procedure CombinarArchivos (ArchivoEntradaA, ArchivoEntradaB, ArchivoSalida)

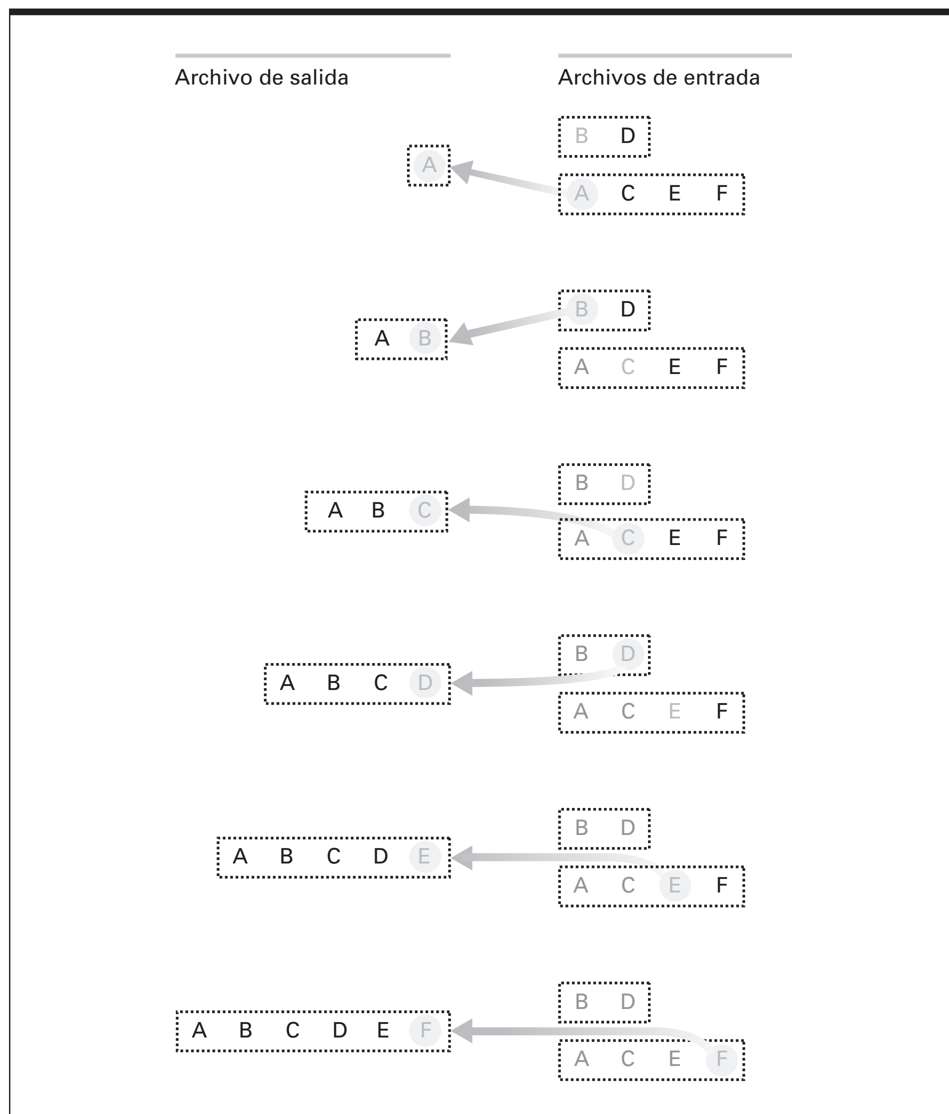
if (ambos archivos de entrada en EOF) then (Parar, con ArchivoSalida vacía)
if (ArchivoEntradaA no está en EOF) then (Declarar su primer registro como registro actual del archivo)
if (ArchivoEntradaB no está en EOF) then (Declarar su primer registro como registro actual del archivo)
while (ninguno de los archivos de entrada está en EOF) do
   (Poner en ArchivoSalida aquel de los dos registros actuales que tenga el “menor” valor en el campo clave;
    if (ese registro actual es el último de su correspondiente archivo de entrada)
       then (Declarar que ese archivo de entrada está en EOF)
       else (Declarar el siguiente registro de ese archivo de entrada como registro actual del archivo)
   )
Comenzando con el registro actual del archivo de entrada que no esté en EOF,
     copiar los registros restantes en ArchivoSalida.
```

orden ascendente. En la Figura 9.15 se resume el algoritmo clásico de combinación. El aspecto que hay que resaltar es que el archivo de salida se construye a medida que se van explorando secuencialmente los dos archivos de entrada (Figura 9.16).

Archivos indexados

Los archivos secuenciales resultan ideales para almacenar datos que van a procesarse en el orden en el que están almacenadas las entradas del archivo. Sin embargo, dichos archivos son poco eficientes cuando hay que extraer los registros del archivo en un orden impredecible. En estas situaciones, lo que hace falta es

Figura 9.16 Aplicación del algoritmo de combinación. (Se utilizan letras para representar registros completos. La letra concreta indica el valor del campo clave del registro.)



una forma de identificar rápidamente la ubicación del registro lógico deseado. Una solución común consiste en utilizar un índice para el archivo, de forma bastante similar a como se emplea un índice en un libro para localizar los temas que el libro contiene. Este tipo de sistema de archivos se denomina **archivo indexado**.

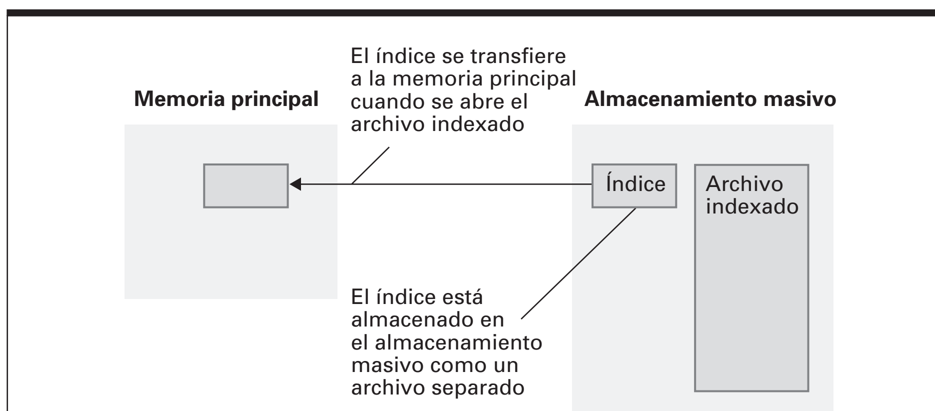
Un índice de un archivo contiene una lista de las claves almacenadas en el archivo, junto con entradas que indican dónde está almacenado el registro que contiene cada clave. Así, para encontrar un registro concreto, lo que hay que hacer es localizar la clave de identificación en el índice y luego extraer el bloque de información almacenado en la ubicación almacenada con dicha clave.

El índice de un archivo suele guardarse como un archivo independiente en el mismo dispositivo de almacenamiento masivo que el archivo indexado. Normalmente, el índice se transfiere a la memoria principal antes de que comience el procesamiento del archivo, para que se pueda acceder a él fácilmente cuando se necesite acceder a los registros del archivo (Figura 9.17).

Un ejemplo clásico de archivo indexado podría ser el que se utiliza en el contexto de mantenimiento de los registros de empleados. En ese caso, podemos utilizar un índice para evitar las largas búsquedas a la hora de extraer un registro individual. En particular, si indexamos el archivo que contiene los registros de empleados por los números de identificación de empleado, entonces podremos extraer rápidamente el registro de un empleado si conocemos su número de identificación. Otro ejemplo lo tenemos en los CD de audio en los que se utiliza un índice para acceder de forma relativamente rápida a cada canción o fragmento individual.

A lo largo de los años, se han utilizado numerosas variantes del concepto básico de índice. Una de esas variantes construye un índice de forma jerárquica de modo que el índice adopta una estructura en capas o de árbol. Un ejemplo prominente sería el sistema de directorios jerárquico utilizado por la mayoría de los sistemas operativos para organizar el almacenamiento de archivos. En ese caso, los directorios, o carpetas, desempeñan el papel de índices, cada uno de los cuales contiene enlaces a sus subíndices. Desde esta perspectiva, el sistema de archivos completo podría considerarse como un único gran archivo indexado.

Figura 9.17 Apertura de un archivo indexado.



Archivos hash

Aunque la indexación proporciona un acceso relativamente rápido a las entradas contenidas en una estructura de almacenamiento de datos, lo hace a expensas de tener que realizar una serie de tareas de mantenimiento del índice. El **hashing** es una técnica que proporciona un acceso similar sin tal sobrecarga. Al igual que en el caso de un sistema indexado, el hashing permite localizar un registro por medio de un valor clave, pero en lugar de buscar la clave en un índice, el hashing identifica la ubicación del registro directamente a partir de la clave.

Un sistema hash puede resumirse de la forma siguiente: se divide el espacio de almacenamiento de datos en varias secciones, denominadas **fragmentos** (*buckets*), cada uno de los cuales es capaz de albergar varios registros. Los registros se dispersan entre los fragmentos de acuerdo con un algoritmo que convierte los valores clave en números de fragmento (esta conversión entre valores clave y números de fragmento se denomina **función hash**). Cada registro se almacena en el fragmento identificado mediante este proceso. De este modo, un registro que haya sido almacenado en la estructura de almacenamiento podrá extraerse aplicando primero la función hash a la clave de identificación del registro, con el fin de determinar el fragmento apropiado y extrayendo después el contenido de dicho fragmento, para finalmente buscar el registro deseado en ese fragmento extraído.

El hashing no se utiliza solo como medio de extraer datos de un sistema de almacenamiento masivo, sino también como medio de extraer grandes bloques de datos almacenados en la memoria principal. Cuando se aplica el hashing a una estructura guardada en un sistema de almacenamiento masivo, el resultado se denomina **archivo hash**. Cuando se aplica a una estructura guardada en la memoria principal, el resultado suele denominarse **tabla hash**.

Autenticación mediante hashing

El hashing es mucho más que un medio de construir sistemas eficientes de almacenamiento de datos. Por ejemplo, el hashing se puede utilizar como medio de autenticar los mensajes transferidos a través de Internet. La técnica consiste en obtener un valor hash del mensaje de una manera secreta. Este valor se transfiere entonces junto con el propio mensaje. Para autenticar el mensaje, el receptor calcula el valor hash del mensaje recibido (de la misma forma secreta) y confirma que el valor generado concuerda con el original. (Asumimos que la probabilidad de que un mensaje alterado tenga el mismo valor hash es muy pequeña.) Si el valor obtenido no concuerda con el original, quedará claro que el mensaje ha sido corrompido. Los lectores interesados en este tema pueden buscar información en Internet acerca de MD5, que es una función hash utilizada ampliamente en aplicaciones de autenticación.

Resulta bastante esclarecedor considerar las técnicas de detección de errores como una aplicación de la técnica de hash a la autenticación. Por ejemplo, el uso de bits de paridad es esencialmente un sistema en el que se aplica una función hash a un patrón de bits para generar un 0 o un 1. Este valor se transfiere posteriormente junto con el patrón original. Si el patrón recibido por el destinatario no genera el mismo valor al aplicarle la función hash se considera que el patrón se ha corrompido.

Apliquemos la técnica de hashing al archivo clásico de empleados en el que cada registro contiene información acerca de un único empleado de la empresa. En primer lugar, establecemos varias áreas disponibles de almacenamiento masivo, que desempeñarán el papel de los fragmentos. El número de fragmentos y el tamaño de cada uno de ellos son decisiones de diseño que luego analizaremos. Por el momento, vamos a suponer que hemos creado 41 fragmentos, a los que denominaremos fragmento número 0, fragmento número 1, hasta el fragmento número 40 (enseguida explicaremos la razón por la que hemos seleccionado 41 fragmentos, en lugar de un número par como 40).

Vamos a suponer que el número de identificación se utiliza como clave para localizar el registro del empleado. Nuestra siguiente tarea entonces consistirá en desarrollar una función hash para convertir estas claves en números de fragmentos. Aunque los "números" de identificación de los empleados puedan tener la forma 25X3Z o J2X35 y sean, por tanto, no numéricos, el hecho es que están almacenados como patrones de bits y podemos interpretar esos patrones de bits como números. Utilizando esta interpretación numérica podemos dividir cualquier clave entre el número de fragmentos disponibles y quedarnos con el resto, que en nuestro caso será un entero comprendido en el rango entre 0 y 40. Por tanto, podemos emplear el resto de este proceso de división para identificar uno de los 41 fragmentos (Figura 9.18).

Utilizando esto como función hash, podemos construir el archivo considerando cada registro individualmente, aplicando nuestra función hash consistente en dividir su clave entre 41 para obtener un número de fragmento y almacenando luego el registro en dicho fragmento (Figura 9.19). Posteriormente, si necesitamos extraer un registro, solo tenemos que aplicar la función hash a la clave con el fin de identificar el fragmento apropiado, para después buscar en ese fragmento el registro en cuestión.

Llegados a este punto vamos a analizar nuestra decisión de dividir el área de almacenamiento en 41 fragmentos. En primer lugar, observe que para obtener un sistema de hash eficiente, los registros que queremos almacenar deben estar distribuidos de manera uniforme entre los fragmentos. Si un número des-

Figura 9.18 Aplicación de la función hash al valor 25X3Z del campo clave, para asignarle uno de los 41 fragmentos.

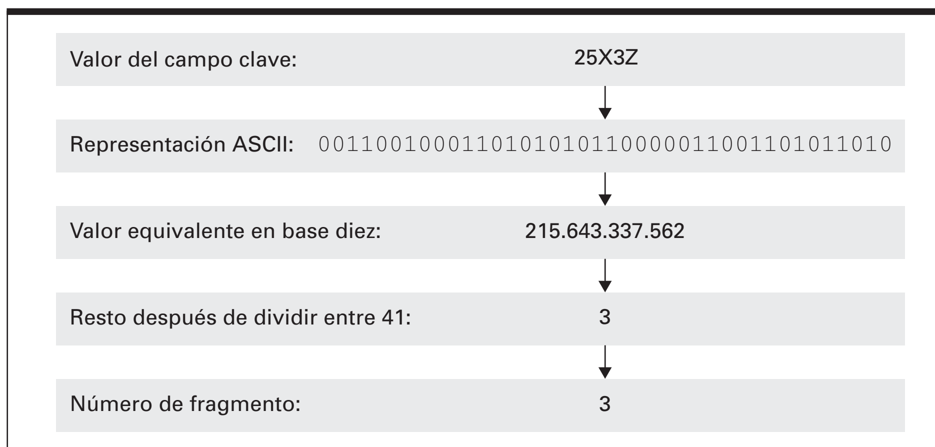
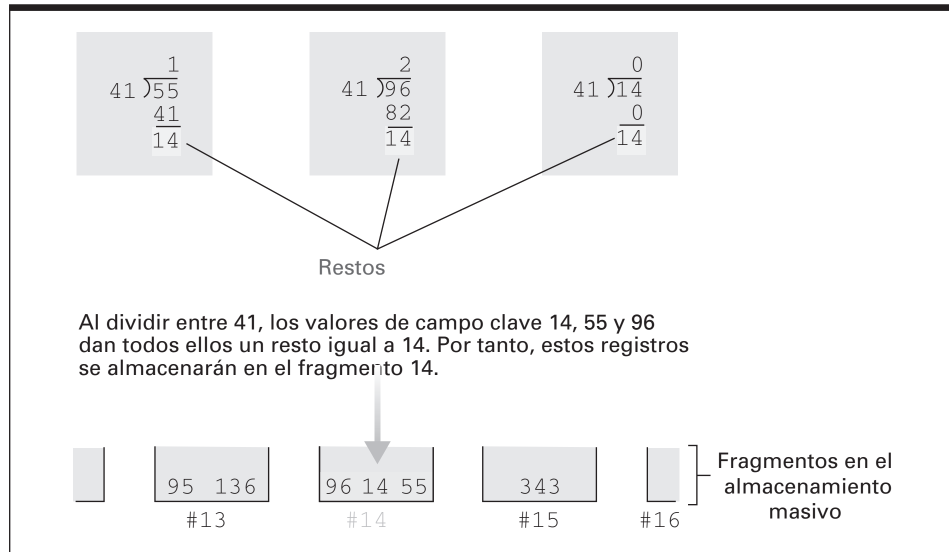


Figura 9.19 Rudimentos de un sistema hash.



proporcionado de claves produjeran, mediante la función hash, el mismo número de fragmento (un fenómeno denominado **agrupación**), entonces habría un número desproporcionado de registros almacenado en un mismo fragmento. A su vez, el extraer un registro de ese fragmento requeriría una búsqueda muy larga, perdiéndose así todas las ventajas que hubiéramos podido ganar mediante la técnica de hash.

Observe ahora que si hubiéramos decidido dividir el área de almacenamiento en 40 fragmentos en lugar de en 41, nuestra función hash hubiera implicado dividir las claves entre el valor 40 en lugar de 41. Pero si un dividendo y un divisor tienen un factor común, dicho factor estará presente también en el resto. En particular, si las claves de las entradas almacenadas en nuestro archivo hash resultaran ser múltiplos de 5 (que es también un divisor de 40), entonces el factor 5 aparecería en los restos al dividir entre 40, y las entradas tenderían a agruparse en aquellos fragmentos que estuvieran asociados con los restos 0, 5, 10, 15, 20, 25, 30 y 35. Una situación similar se produciría en el caso de claves que fueran múltiplos de 2, 4, 8, 10 y 20, porque todas ellas son también factores de 40. En consecuencia, hemos decidido dividir el área de almacenamiento en 41 fragmentos, porque la elección de 41, al ser un número primo, elimina la posibilidad de que existan factores comunes y reduce por tanto la probabilidad de que se produzcan agrupamientos.

Lamentablemente, nunca puede eliminarse del todo la posibilidad de agrupamiento. Incluso con una función hash bien diseñada es extremadamente probable que dos claves tengan el mismo valor hash (un fenómeno denominado **colisión**) en las etapas tempranas de construcción del archivo. Para entender por qué, considere el siguiente escenario.

Suponga que hemos encontrado una función hash que distribuye arbitrariamente los registros entre 41 fragmentos. Suponga también que nuestro sistema de almacenamiento está vacío y que vamos a insertar nuevos registros de

uno en uno. Al insertar el primer registro se colocará en un segmento vacío. Sin embargo, al insertar el siguiente solo seguirán vacíos 40 de los 41 fragmentos, por lo que la probabilidad de que el segundo registro se almacene en un fragmento vacío es solo de 40/41. Suponiendo que el segundo registro se almacene en un fragmento vacío, el tercer registro solo encontrará 39 fragmentos vacíos, por lo que la probabilidad de que se almacene en uno de ellos será de 39/41. Continuando con este proceso, vemos que si los primeros siete registros se almacenan en fragmentos vacíos, el octavo fragmento tendrá solo una probabilidad de 34/41 de ser guardado en uno de los fragmentos vacíos restantes.

Este análisis nos permite calcular la probabilidad de que los primeros ocho registros queden todos ellos guardados en fragmentos vacíos: esa probabilidad será el producto de las probabilidades de que cada registro se almacene en un fragmento vacío, asumiendo que también las entradas precedentes fueran almacenadas en un fragmento vacío. Esta probabilidad es igual a

```
(41/41)(40/41)(39/41)(38/41) . . . (34/41) = 0,482
```

Como vemos, el resultado es inferior a un medio; es decir, al distribuir registros entre 41 fragmentos, la probabilidad de que se haya producido ya una colisión en el momento de almacenar el octavo registro es mayor que la probabilidad de que no se haya producido.

La alta probabilidad de las colisiones indica que, independientemente de lo bien que se elija una función hash, es preciso diseñar los sistemas hash teniendo presente el fenómeno del agrupamiento. En particular, es posible que un fragmento termine por llenarse y se desborde. Una solución a este problema sería permitir que los fragmentos se expandan en tamaño. Otra solución es permitir que los fragmentos desborden sus registros sobrantes en un área de desbordamiento que se haya reservado para ese propósito. En cualquiera de los casos, los fenómenos de agrupamiento y de desbordamiento de fragmentos pueden degradar significativamente el rendimiento de un archivo hash.

Las investigaciones demuestran que, como regla general, los archivos hash se comportan bien siempre que el cociente entre el número de registros y la capacidad total de registros del archivo (un cociente que se conoce como **factor de carga**) permanezca por debajo del 50 por ciento. Sin embargo, en cuanto el factor de carga comienza a superar el 75 por ciento, el rendimiento del sistema suele degradarse (aparece el fenómeno del agrupamiento, haciendo que algunos fragmentos se llenen y posiblemente se desborden). Por esta razón, los sistemas de almacenamiento hash suelen reconstruirse con una capacidad mayor en cuanto su factor de carga se aproxima al 75 por ciento. Como conclusión, podemos decir que la eficiencia en la extracción de registros que se obtiene al implementar un sistema hash tiene sus costes de implementación asociados.

Cuestiones y ejercicios

1. Siga el algoritmo de combinación presentado en la Figura 9.15, asumiendo que uno de los archivos de entrada contiene registros cuyos valores de campo clave son B y E , mientras que el otro contiene A, C, D y F.

2. El algoritmo de combinación es la base de un conocido algoritmo de ordenación que se denomina ordenación por combinación. Describa cuál podría ser este algoritmo. (***Sugerencia:*** cualquier archivo no vacío puede considerarse como un conjunto de archivos de una única entrada.)
3. ¿El carácter de secuencial es una propiedad física o conceptual de un archivo?
4. ¿Cuáles son los pasos requeridos a la hora de extraer un registro de un archivo indexado?
5. Explique por qué una función hash mal elegida puede provocar que un sistema de almacenamiento hash sea poco más que un archivo secuencial.
6. Suponga que construimos un sistema de almacenamiento hash utilizando la función de división hash tal como la hemos presentado en el texto, pero con seis fragmentos de almacenamiento. Para cada uno de los siguientes valores clave, identifique el fragmento en el que se almacenaría el registro que tenga dicha clave. ¿Cuál es el problema existente y por qué?

   a. 24 **b.** 30 **c.** 3 **d.** 18 **e.** 15
   f. 21 **g.** 9 **h.** 39 **i.** 27 **j.** 0
7. ¿Cuántas personas habrá que juntar antes de que la probabilidad de que dos miembros del grupo cumplan años el mismo día sea mayor del 50 por ciento? ¿Cómo se relaciona este problema con los conceptos presentados en esta sección?

9.6 Minería de datos

Un campo en rápida expansión que está estrechamente relacionado con la tecnología de bases de datos es el de la minería de datos, que son técnicas que permiten descubrir patrones dentro de conjuntos de datos. La minería de datos se ha convertido en una herramienta de gran importancia en numerosas áreas, incluyendo el marketing, la gestión de inventarios, el control de calidad, la gestión de riesgos crediticios, la detección de fraudes y el análisis de inversiones. Las técnicas de minería de datos tienen incluso aplicación en entornos tan improbables como la identificación de las funciones de determinados genes codificados en las moléculas de ADN y la caracterización de propiedades de los organismos.

Las actividades de la minería de datos difieren de las consultas tradicionales a una base de datos en que la minería de datos trata de identificar patrones previamente desconocidos por oposición a las consultas tradicionales, que simplemente solicitan que se extraigan hechos ya almacenados. Además, la minería de datos se practica con colecciones de datos estáticas, denominadas **almacenes de datos**, en lugar de con bases de datos operacionales “en línea”, que están sujetas a actualizaciones frecuentes. Estos almacenes de datos suelen ser “instantáneas” de bases de datos o de conjuntos de bases de datos. Se utilizan en lugar de las bases de datos operacionales reales porque la localiza-

ción de patrones en un sistema estático es mucho más sencilla que en un sistema dinámico.

También debemos observar que el tema de la minería de datos no está restringido al dominio de la computación, sino que sus tentáculos se extienden bastante más alla para entrar en el campo de la estadística. De hecho, muchos argumentarían que puesto que la minería de datos tiene su origen en los intentos de realizar análisis estadísticos sobre grandes conjuntos de datos diversos es una aplicación de la estadística, más que un campo de las Ciencias de la computación.

Dos formas muy comunes de la minería de datos son la **descripción de clases** y la **discriminación de clases**. La descripción de clases trata de identificar propiedades que caractericen a un grupo dado de elementos de datos, mientras que la discriminación de clases trata de identificar propiedades que separen a dos grupos. Por ejemplo, las técnicas de descripción de clases se utilizarían para identificar las características de las personas que han comprado vehículos pequeños y baratos, mientras que las técnicas de discriminación de clases se emplearían para encontrar las propiedades que distinguen a los clientes que compran vehículos usados de aquellos que compran vehículos de primera mano.

Otra forma de minería de datos es el **análisis de agrupamiento**, que trata de descubrir clases. Observe que esta técnica difiere de la de descripción de clases, que intenta descubrir propiedades de los miembros pertenecientes a clases que ya han sido identificadas. Para ser más precisos, el análisis de agrupamiento trata de encontrar propiedades de los elementos de datos que conduzcan al descubrimiento de grupos. Por ejemplo, al analizar la información acerca de las edades de las personas que han visto una película concreta, el análisis de agrupamiento podría descubrir que la base de espectadores se descompone en dos grupos de edad: un grupo de edad de 4 a 10 años y otro grupo de 25 a 40 años (¿quizá esa película atrajo a los niños y a sus padres?).

Otra forma más de minería de datos es el **análisis de asociaciones**, que implica la búsqueda de enlaces entre grupos de datos. Es el análisis de asociaciones el que podría revelar, por ejemplo, que los clientes que compran patatas fritas también compran cerveza y refrescos, o que las personas que hacen sus compras durante las horas de trabajo normales también cobran una pensión de algún organismo público.

El **análisis de excepciones** es otra forma de minería de datos. Esta técnica trata de identificar aquellos elementos de datos que no se ajustan a la norma. El análisis de excepciones puede utilizarse para localizar errores en los conjuntos de datos, para detectar el robo de tarjetas de crédito observando las desviaciones súbitas con respecto a los patrones normales de compra de un cliente y quizá para identificar terroristas potenciales detectando comportamientos inusuales.

Por último, la forma de minería de datos denominada **análisis de patrones secuenciales** trata de identificar patrones de comportamiento a lo largo del tiempo. Por ejemplo, el análisis de patrones secuenciales podría revelar tendencias implícitas en los sistemas económicos, como por ejemplo los mercados de valores, o en sistemas medioambientales, como por ejemplo las condiciones climáticas.

Bioinformática

Los avances en la tecnología de bases de datos y en las técnicas de minería de datos están ampliando el repertorio de herramientas que los biólogos tienen a su disposición en las áreas de investigación que implican la identificación de patrones y la clasificación de compuestos orgánicos. El resultado es un nuevo campo de la Biología denominado Bioinformática. Teniendo su origen en los esfuerzos para decodificar el ADN, la Bioinformática abarca ahora tareas tales como la catalogación de proteínas y los intentos de comprender las secuencias de interacción proteínica (denominadas rutas bioquímicas). Aunque normalmente se considera parte de la Biología, la Bioinformática es un ejemplo de cómo las Ciencias de la computación están influyendo en otros campos e incluso integrándose en ellos.

Como indica este último ejemplo, los resultados de la minería de datos pueden utilizarse para predecir comportamientos futuros. Si una entidad posee las propiedades que caracterizan a una clase, entonces esa entidad se comportará probablemente como los miembros de esa clase. Sin embargo, muchos proyectos de minería de datos tratan simplemente de comprender mejor datos ya existentes, como lo atestigua el uso de la minería de datos a la hora de desentrañar los misterios del ADN. En cualquier caso, el ámbito de posibles aplicaciones de la minería de datos es enorme, así que este campo promete constituir un área activa de investigación durante los años venideros.

Observe que la tecnología de bases de datos y la de minería de datos están emparentadas, por lo que las investigaciones en uno de los dos campos tendrá repercusiones en el otro. Las técnicas de bases de datos se utilizan ampliamente para proporcionar a los almacenes de datos la capacidad de presentar la información en forma de **cubos de datos** (datos contemplados desde múltiples perspectivas, el término *cubo* se utiliza para evocar la imagen de múltiples dimensiones) y es esa capacidad de presentación de la información lo que hace posible la minería de datos. A su vez, a medida que los investigadores en la minería de datos mejoran las técnicas para implementar cubos de datos, estos resultados irán sirviendo para mejorar el campo del diseño de bases de datos.

Para terminar, es preciso reconocer que una adecuada aplicación de la minería de datos abarca mucho más que la identificación de patrones dentro de un conjunto de datos. Es preciso aplicar un juicio inteligente para determinar si esos patrones son significativos o simples coincidencias. El hecho de que una administración concreta de lotería haya vendido un alto número de boletos premiados probablemente no debería considerarse como significativo para alguien que está planeando comprar un billete de lotería, pero el descubrimiento de que los clientes que compran bolsitas de aperitivos también tienden a adquirir alimentos congelados sí que podría constituir una información significativa para los gerentes de supermercados. De la misma forma, la minería de datos abarca un amplio rango de cuestiones éticas, relativas a los derechos de los individuos representados en el almacén de datos, a la precisión y el uso de las conclusiones que se extraigan e incluso si la propia minería de datos es aceptable.

Cuestiones y ejercicios

1. ¿Por qué la minería de datos no se lleva a cabo con bases de datos "en línea"?
2. Proporcione un ejemplo adicional de patrón que podría encontrarse para cada uno de los tipos de minería de datos identificados en el texto.
3. Identifique algunas de las diferentes perspectivas que podrían utilizarse con un cubo de datos a la hora de aplicar técnicas de minería de datos a la información relativa a las ventas de una empresa.
4. ¿En qué difiere la minería de datos de las consultas tradicionales de bases de datos?

9.7 Impacto social de la tecnología de bases de datos

Con el desarrollo de la tecnología de las bases de datos, esa información que antaño estaba enterrada en misteriosos archivos ha pasado a ser accesible. En muchos casos, los sistemas automatizados de gestión de bibliotecas hacen que resulte fácil conocer los hábitos de lectura de los usuarios; los vendedores minoristas mantienen registros de las compras efectuadas por sus clientes y los motores de búsqueda en Internet también conservan memoria de las solicitudes que los clientes realizan. A su vez, esta información está potencialmente disponible para las empresas de marketing, las fuerzas de seguridad, los partidos políticos, los empresarios y las propias personas individuales.

Esto es bastante representativo de los problemas potenciales que presenta todo el espectro existente de aplicaciones de bases de datos. La tecnología ha hecho que sea fácil recopilar cantidades enormes de datos y combinar o comparar diferentes conjuntos de datos con el fin de detectar relaciones que de otro modo permanecerían ocultas bajo la avalancha de información. Las ramificaciones de esto, tanto positivas como negativas, son enormes. Y no son un simple objeto de debate académico, sino que son una auténtica realidad.

En algunos casos, el proceso de recopilación de datos se hace de manera abierta; en otros, se trata de una actividad mucho más sutil. Como ejemplos del primer caso podemos citar aquellos en los que se nos pide explícitamente que proporcionemos información. Esa información puede ser suministrada de manera voluntaria, como sucede en las encuestas o en los formularios de registro para un concurso, o puede ser aportada de manera involuntaria, como cuando nos es impuesta por las leyes existentes. En ocasiones, el que sea voluntario o no depende de nuestro punto de vista. ¿Proporcionar información personal a la hora de solicitar un crédito es un hecho voluntario o involuntario? La diferencia está en si recibir el crédito es una necesidad o se trata, por el contrario, de algo prescindible. Para utilizar una tarjeta de crédito ahora se requiere en muchos sitios que permitamos que nuestra firma se archive en formato digitalizado. De nuevo, el proporcionar esa información puede considerarse algo voluntario o involuntario, dependiendo de la situación de cada cual.

Los otros casos más sutiles de recopilación de datos evitan la comunicación directa con la persona. Por ejemplo, las empresas de tarjetas de crédito, que

registran los hábitos de compra de los poseedores de sus tarjetas, los sitios web que registran las identidades de aquellos que visitan el sitio o los activistas que anotan los números de matrícula de los vehículos que se encuentran en el aparcamiento de algunas de las instituciones que son objeto de su ira. En estos casos, la persona que está siendo sometida a ese proceso de recopilación de datos puede no ser consciente de que esa información se está recopilando, y es probable que sea aún menos consciente de la existencia de esas bases de datos que se construyen a partir de dicha información.

En ocasiones, las actividades subyacentes de recopilación de datos son evidentes por sí mismas, solo con que uno se pare a pensar durante unos momentos. Por ejemplo, una tienda puede ofrecer descuentos a sus clientes habituales, exigiéndoles que proporcionen de antemano a la tienda sus datos personales. El proceso de registro puede implicar el entregar al cliente algún tipo de tarjeta de identificación, que luego habrá que presentar en el momento de la compra, para poder obtener el descuento. El resultado es que esa tienda será capaz de compilar un registro de los hábitos de compra de los clientes. Un registro cuyo valor es mucho mayor que el de los descuentos concedidos.

Por supuesto, la fuerza que impulsa este florecimiento de los procesos de recopilación de datos es, precisamente, el valor de esos datos, que se ve amplificado por los avances en la tecnología de bases de datos, los cuales permiten enlazar esos datos de distintas maneras con el fin de descubrir información que de otro modo permanecería oculta. Por ejemplo, los hábitos de compra de los poseedores de tarjetas de crédito pueden clasificarse y compararse, con el fin de obtener perfiles de cliente que tienen un valor inmenso en las tareas de marketing. Los formularios de subscripción para revistas de culturismo pueden enviarse a aquellos que hayan adquirido recientemente aparatos de gimnasia, mientras que los formularios de subscripción a revistas sobre entrenamiento de perros pueden dirigirse a aquellos que hayan comprado en las últimas semanas comida para perros. Las distintas formas de combinar la información son, en ocasiones, muy imaginativas. Por ejemplo, en Estados Unidos se comparan los registros de asistencia social con los registros policiales, con el fin de localizar y detener a aquellos que han violado la libertad condicional y, en 1984, el gobierno americano llegó a utilizar antiguas listas de registro con fechas de cumpleaños, de un popular fabricante de helados con el fin de identificar a aquellos ciudadanos que no se habían registrado para hacer el servicio militar.

Hay distintos enfoques para proteger a la sociedad del uso abusivo de las bases de datos. Uno es aplicar remedios legales. Lamentablemente, el que se apruebe una ley contra un determinado tipo de actuación, no impide que esa actuación se produzca, sino solo la hace ilegal. Un ejemplo sobresaliente en Estados Unidos es la Ley de Intimidad de 1974, cuyo objetivo era proteger a los ciudadanos del uso abusivo de las bases de datos gubernamentales. Una de las medidas de esta ley requería que los organismos gubernamentales publicitaran la existencia de sus bases de datos mediante un registro federal que permitiera a los ciudadanos acceder a su información personal y corregirla. Sin embargo, los organismos gubernamentales tardaron mucho en ajustarse a esta norma. Esto no implica necesariamente que hubiera una intención maliciosa: en muchos casos, el problema era la burocracia. Pero el simple hecho de que un sistema burocrático pueda estar construyendo bases de datos con información personal que luego es incapaz de publicitar no es precisamente tranquilizador.

Otra técnica, quizá más potente, para controlar el abuso de las bases de datos es la opinión pública. Nadie abusaría de las bases de datos si el castigo fuera superior a los beneficios que se obtienen. Y el castigo que las empresas temen más es precisamente una opinión pública adversa, porque puede repercutir directamente en los resultados de la empresa. A principios de la década de 1990, fue la opinión pública la que terminó por hacer que las empresas de tarjetas de crédito desistieran de vender listas de correo para propósitos de marketing. Más recientemente, America Online (uno de los principales proveedores de servicios Internet) renunció por presiones de la opinión pública a su política de vender información relativa a sus clientes a empresas de telemarketing. Ha habido incluso organismos gubernamentales que han tenido que plegarse ante las críticas de la opinión pública. En 1997, la Administración de la Seguridad Social de Estados Unidos modificó sus planes de hacer disponibles a través de Internet los registros de los afiliados a la Seguridad Social, cuando la opinión pública cuestionó la seguridad de esa información. En esos casos, se obtuvieron resultados en cuestión de días, lo que contrasta enormemente con los dilatados periodos de tiempo que suelen asociarse con los procesos legales.

Por supuesto, en muchos casos las aplicaciones de bases de datos son beneficiosas tanto para el que posee los datos como para el que los proporciona, aunque siempre existe una pérdida de intimidad que no hay que tomarse a la ligera. Tales problemas de intimidad son graves cuando la precisión de la información es adecuada, pero se hacen realmente enormes cuando, encima, la información es errónea. Imagine la sensación de desesperanza cuando uno se da cuenta de que le han rebajado el límite de sus tarjetas de crédito debido a algún tipo de información errónea. E imagine después cómo se multiplicarían nuestros problemas en un entorno en el que esa información errónea fuera compartida fácilmente con otras instituciones.

Los problemas de intimidad son uno de los mayores efectos colaterales de los avances de la tecnología en general y de las técnicas de bases de datos en particular. Y continuarán representando un grave problema en el próximo futuro. Las soluciones a estos problemas exigen que los ciudadanos se informen, estén permanentemente alerta y sean activos.

Cuestiones y ejercicios

1. ¿Debería concederse a las fuerzas de seguridad acceso a bases de datos con el propósito de identificar a personas con tendencias criminales, aún cuando esas personas pueden no haber cometido ningún crimen?
2. ¿Debería concederse a las empresas de seguros acceso a bases de datos para identificar a personas que tengan problemas médicos potenciales, aún cuando esas personas no hayan mostrado ningún síntoma?
3. Suponga que se encuentra en una situación financiera desahogada. ¿Qué ventajas podría obtener si esta información fuera compartida por diversas instituciones? ¿Qué desventajas podría traerle la distribución de esa misma información? ¿Y en el caso de que su situación financiera no fuera desahogada?

4. ¿Qué papel tiene la prensa libre a la hora de controlar el abuso de bases de datos? (Por ejemplo, ¿en qué grado contribuye la prensa a la formación de la opinión pública o a poner de manifiesto los abusos?)

Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. Resuma las diferencias entre un archivo plano y una base de datos.

2. ¿Qué queremos decir al hablar de independencia de los datos?

3. ¿Cuál es el papel de un sistema DBMS en la arquitectura de niveles de una implementación de base de datos?

4. ¿Cuál es la diferencia entre un esquema y un subesquema?

5. Identifique dos ventajas de separar el software de aplicación del DBMS.

6. Describa la similitudes entre un tipo abstracto de datos (Capítulo 8) y un modelo de base de datos.

7. Identifique el nivel dentro de un sistema de base de datos (usuario, programador del software de aplicación, diseñador del software DBMS) en el que tiene lugar cada una de estas actividades o en el que se presenta cada uno de estos problemas:
   a. ¿Cómo habría que almacenar los datos en un disco para maximizar la eficiencia?
   b. ¿Hay algún asiento libre en el vuelo 243?
   c. ¿Cómo debe organizarse una relación en el almacenamiento masivo?
   d. ¿Cuántas veces hay que permitir que un usuario escriba mal su contraseña antes de dar por terminada la conversación?
   e. ¿Cómo puede implementarse la operación PROJECT?

8. ¿Cuáles de las siguientes tareas son gestionadas por un DBMS?
   a. Garantizar que el acceso de un usuario a la base de datos está restringido al subesquema apropiado.
   b. Traducir los comandos expresados en términos del modelo de base de datos a acciones compatibles con el sistema real de almacenamiento de datos.
   c. Ocultar el hecho de que los datos de la base de datos en la práctica están dispersos entre múltiples computadoras conectadas en red.

9. Describa cómo se representaría en una base de datos relacional la siguiente información acerca de líneas aéreas, vuelos (para un día concreto) y pasajeros:

Líneas aéreas: Clear Sky, Long Hop y Tree Top

Vuelos de Clear Sky: CS205, CS37 y CS102

Vuelos de Hop: LH67 y LH89

Vuelos de Tree Top: TT331 y TT809

Suarez tiene reservas en los vuelos CS205 (asiento 12B), CS37 (asiento 18C) y LH 89 (asiento 14A).

Barrios tiene reservas en los vuelos CS37 (asiento 18B) y LH89 (asiento 14B).

Salas tiene reservas en los vuelos LH67 (asiento 5A) y TT331 (asiento 4B).

10. ¿Hasta qué punto es significativo el orden en que se aplican las operaciones SELECT y PROJECT a una relación? Es decir, ¿en qué condiciones el seleccionar y luego proyectar producirá los mismos resultados que si primero proyectamos y luego seleccionamos?

11. Proporcione un argumento que demuestre que la cláusula “where” de la operación JOIN, tal como se describe en la Sección 9.2, no es necesaria. (Es decir, demuestre que

cualquier consulta que utilice una cláusula "where" puede expresarse también utilizando una operación JOIN que concatene cada tupla de una relación con cada tupla de la otra.)

12. En función de las relaciones que se muestran a continuación, ¿cuál sería el aspecto de la relación RESULT después de ejecutar cada una de estas sentencias:

a. RESULT ← PROJECT U and W from X1
b. RESULT ← SELECT from X1 where W = 50
c. RESULT ← PROJECT S from X2
d. RESULT ← JOIN X1 and X2 where X1.W ≥ X2.R

relación X1

U	V	W
AA	ZZ	50
BB	DD	30
CC	QQ	50

relación X2

R	S
30	JJ
40	KK

13. Utilizando los comandos SELECT, PROJECT y JOIN, escriba una secuencia de instrucciones para responder a cada una de las siguientes cuestiones acerca de los componentes y de sus fabricantes en términos de la siguiente base de datos:

relación COMPONENTE

NombreComp	Peso
Tornillo 2X	1
Tornillo 2Z	1.5
Tuerca V5	0.5

relación FABRICANTE

NombreEmpresa	NombreComp	Coste
Empresa X	Tornillo 2Z	.03
Empresa X	Tuerca V5	.01
Empresa Y	Tornillo 2X	.02
Empresa Y	Tuerca V5	.01
Empresa Y	Tornillo 2Z	.04
Empresa Z	Tuerca V5	.01

a. ¿Qué empresas fabrican el tornillo 2Z?
b. Obtenga una lista de los componentes fabricados por la Empresa X junto con el coste de cada componente.
c. ¿Qué empresas fabrican algún componente con un peso igual a 1?

14. Responda al Problema 13 utilizando SQL.

15. Utilizando los comandos SELECT, PROJECT y JOIN, escriba secuencias que permitan responder a las siguientes cuestiones acerca de las relaciones EMPLEADO, PUESTO y ASIGNACION de la Figura 9.5:

a. Obtenga una lista de los nombres y direcciones de los empleados de la empresa.
b. Obtenga una lista de los nombres y direcciones de aquellos que han trabajado o están trabajando en el departamento de personal.
c. Obtenga una lista de los nombres y direcciones de aquellos que están trabajando en el departamento de personal.

16. Responda al problema anterior utilizando SQL.

17. Diseñe una base de datos relacional que contenga información acerca de compositores de música, sus vidas y sus obras. (Evite redundancias similares a las de la Figura 9.4.)

18. Diseñe una base de datos relacional que contenga información acerca de autores, de sus obras y de sus editores. (Evite redundancias similares a las de la Figura 9.4.)

19. Diseñe una base de datos relacional que contenga información acerca de una editorial y de los títulos que ha publicado. (Evite redundancias similares a las de la Figura 9.4.)

20. Diseñe una base de datos relacional que contenga información acerca de empresas, productos y clientes. (Evite redundancias similares a las de la Figura 9.4.)

21. Diseñe una base de datos relacional que contenga información acerca de los empleados de una empresa, de sus proyectos y de la distribución de los empleados para los proyectos. En este sistema, cada empleado puede trabajar en más de un proyecto y un proyecto puede tener más de un empleado trabajando en él. (Evite redundancias similares a las de la Figura 9.4.)

22. Escriba una secuencia de instrucciones (utilizando las operaciones SELECT, PROJECT y

JOIN) para extraer los campos `IdPuesto`, `FechaInicio` y `FechaFin` para cada puesto del departamento de contabilidad a partir de la base de datos relacional descrita en la Figura 9.5.

23. Responda al problema anterior utilizando SQL.

24. Escriba una secuencia de instrucciones (utilizando las operaciones SELECT, PROJECT y JOIN) para extraer los campos `Nombre`, `NSS`, `TitPuesto` y `CodCat` de cada uno de los empleados actuales a partir de la base de datos relacional descrita en la Figura 9.5.

25. Responda al problema anterior utilizando SQL.

26. Escriba una secuencia de instrucciones (utilizando las operaciones SELECT, PROJECT y JOIN) para extraer los campos `Nombre` y `NSS` del empleado cuyo `IdPuesto` es S25X a partir de la base de datos relacional descrita en la Figura 9.5.

27. Responda al problema anterior utilizando SQL.

28. ¿Cuál es la diferencia entre la información suministrada por la única relación

Nombre	Departamento	NumTelefono
Juarez	Ventas	555-2222
Salas	Ventas	555-3333
Barrio	Personal	555-4444

y las dos relaciones

Nombre	Departamento
Juarez	Ventas
Salas	Ventas
Barrio	Personal

Departamento	NumTelefono
Ventas	555-2222
Ventas	555-3333
Personal	555-4444

29. Diseñe una base de datos relacional que contenga información acerca de componentes de automóviles y sus subcomponentes. Asegúrese de permitir que un componente pueda contener componentes más pequeños y, al mismo tiempo, estar contenido en otros componentes más grandes.

30. Elija un sitio web popular, como por ejemplo www.rediffmail.com, www.yahoo.com o www.amazon.com y diseñe una base de datos relacional que pudiera servir como base de datos de soporte para el sitio.

31. En referencia a la base de datos representada en la Figura 9.5, indique a qué cuestión se respondería mediante el siguiente segmento de programa:

```
TEMP ← SELECT from ASIGNACION
  where FechaFin = "*"
RESULT ← PROJECT IdPuesto,
  FechaInicio from TEMP
```

32. Traduzca la consulta del problema anterior a SQL.

33. En referencia a la base de datos representada en la Figura 9.5, indique a qué cuestión se respondería mediante el siguiente segmento de programa:

```
TEMP1 ← JOIN EMPLEADO and ASIGNACION
  where EMPLEADO.IdEmpl =
  ASIGNACION.IdEmpl
TEMP2 ← SELECT from TEMP1 where
  FechaFin = "*"
RESULT ← PROJECT Nombre, FechaInicio
  from TEMP2
```

34. Traduzca la consulta del problema anterior a SQL.

35. En referencia a la base de datos representada en la Figura 9.5, indique a qué cuestión se respondería mediante el siguiente segmento de programa:

```
TEMP1 ← JOIN EMPLEADO and ASIGNACION
  where EMPLEADO.IdEmpl =
            ASIGNACION.IdEmpl
TEMP2 ← SELECT from TEMP1 where
  IdPuesto = "F5"
RESULT ← PROJECT Nombre from TEMP2
```

36. Traduzca la consulta del problema anterior a SQL.

37. Traduzca la siguiente sentencia SQL

```
select PUESTO.TitPuesto
from EMPLEADO, PUESTO
where EMPLEADO.IdPuesto =
         PUESTO.IdPuesto
  and EMPLEADO.IdEmpl = "12Y34"
```

a una secuencia de operaciones SELECT, PROJECT y JOIN.

38. Traduzca la siguiente sentencia SQL

```
select PROYECTO.FechaInicio
from PROYECTO, ESTUDIANTE
where PROYECTO.IdEstudiante =
    ESTUDIANTE.IdEstudiante
  and ESTUDIANTE.Nombre = "David
    Rayo"
```

a una secuencia de operaciones SELECT, PROJECT y JOIN.

39. Describa el efecto que la siguiente sentencia SQL tendría sobre la base de datos del Problema 13.

```
insert into FABRICANTE
values (' Empresa Z','Tornillo 2X',
         .03)
```

40. Describa el efecto que la siguiente sentencia SQL tendría sobre la base de datos del Problema 13.

```
update FABRICANTE
  set Coste = .03
  where NombreEmpresa = 'Empresa Y'
    and NombreComp = 'Tornillo 2X'
```

***41.** Identifique algunos de los objetos que podríamos esperar encontrar en una base de datos orientada a objetos que se utilizara para mantener el inventario de una frutería. ¿Qué métodos esperaría encontrar en cada uno de esos objetos?

***42.** Identifique algunos de los objetos que podríamos esperar encontrar en una base de datos orientada a objetos utilizada para mantener los registros de los libros de una biblioteca. ¿Qué métodos esperaría encontrar en cada uno de esos objetos?

***43.** ¿Qué información incorrecta sería generada por la siguiente secuencia de transacciones T1 y T2?

T1 está diseñada para calcular la suma de las cuentas bancarias A y B; T2 está diseñada para transferir 100 euros de la cuenta A a la B. T1 comienza extrayendo el saldo de la cuenta A; a continuación, T2 realiza su transferencia y, finalmente, T1 extrae el saldo de la cuenta B e informa de la suma de los valores que ha extraído.

***44.** Explique cómo podría resolverse el error existente en el Problema 43 utilizando el protocolo de bloqueo descrito en el texto.

***45.** ¿Qué efecto tendría el protocolo de desalojo y espera sobre la secuencia de sucesos del Problema 43, si T1 fuera la transacción más reciente? ¿Y si la transacción más reciente fuera T2?

***46.** Suponga que una transacción trata de sumar 100 euros a una cuenta bancaria cuyo saldo es de 200 euros, al mismo tiempo que otra transacción trata de retirar 100 euros de la misma cuenta. Describa un entremezclado de estas transacciones que conduciría a un saldo final de 100 euros. Describa un entremezclado de estas transacciones que conduciría a un saldo final de 300 euros.

***47.** ¿Cuál es la diferencia entre una transacción con acceso exclusivo y otra con acceso compartido a un elemento de la base de datos y por qué esa diferencia es importante?

***48.** Los problemas expuestos en la Sección 9.4 en relación con las transacciones concurrentes no están limitados a los entornos de bases de datos. ¿Qué problemas similares surgirían a la hora de acceder a un documento de un procesador de textos? (Si dispone de un PC con un procesador de textos, intente acceder al mismo documento con dos activaciones de ese programa procesador de textos y vea lo que sucede.)

***49.** Suponga que un archivo secuencial contiene 50.000 registros y que hacen falta 5 milisegundos para consultar cada entrada. ¿Cuánto estima que tendríamos que esperar a la hora de extraer un registro situado en mitad del archivo?

***50.** Enumere los pasos que se ejecutan en el algoritmo de combinación de la Figura 9.15 si uno de los archivos de entrada está vacío al comienzo.

***51.** Modifique el algoritmo de la Figura 9.15 para manejar el caso en el que ambos archivos de entrada contengan un registro con el mismo valor en el campo clave. Suponga que estos registros son idénticos y que solo uno de ellos debe aparecer en el registro de salida.

***52.** Diseñe un sistema por el que pueda procesarse como archivo secuencial un archivo almacenado en un disco, con una de dos posibles ordenaciones.

***53.** Describa cómo podría construirse un archivo secuencial que contenga información acerca de los subscriptores de una revista, utilizando un archivo de texto como estructura subyacente.

***54.** Diseñe una técnica que permita implementar como archivo de texto un archivo secuencial cuyos registros lógicos no tengan todos el mismo tamaño. Por ejemplo, suponga que deseamos construir un archivo secuencial en el que cada registro lógico contenga información acerca de un novelista, así como una lista de las obras de dicho autor.

***55.** ¿Qué ventajas tiene un archivo indexado con respecto a un archivo hash? ¿Qué ventajas tiene un archivo hash con respecto a un archivo indexado?

***56.** En el capítulo se ha trazado un paralelismo entre él índice de un archivo tradicional y el sistema de directorios mantenido por un sistema operativo. ¿En qué se diferencian un índice tradicional y el directorio de archivos de un sistema operativo?

***57.** Si partimos un archivo hash en 10 fragmentos, ¿cuál es la probabilidad de que al menos dos de tres registros seleccionados arbitrariamente correspondan al mismo fragmento? (Suponga que la función hash no da a ningún fragmento prioridad con respecto a los otros.) ¿Cuántos registros habrá que almacenar en el archivo hasta para que sea más probable que se produzcan colisiones que no se produzcan?

***58.** Resuelva el problema anterior, suponiendo que el archivo se parte en 100 fragmentos en lugar de en solo 10.

***59.** Si estamos utilizando la técnica de división explicada en el capítulo para obtener una función hash y dividimos el área de almacenamiento del archivo en 23 fragmentos, ¿en qué fragmento deberemos buscar para encontrar el registro cuya clave, al ser interpretada como un valor binario, tenga el entero 124?

***60.** Compare la implementación de un archivo hash con la de una matriz homogénea bidimensional. ¿En qué sentido son similares los papeles de la función hash y del polinomio de dirección?

***61.** Indique una ventaja que tenga
a. un archivo secuencial con respeto a un archivo indexado.
b. un archivo secuencial con respeto a un archivo hash.
c. un archivo indexado con respeto a un archivo secuencial.
d. un archivo indexado con respeto a un archivo hash.
e. un archivo hash con respeto a un archivo secuencial.
f. un archivo hash con respeto a un archivo indexado.

***62.** ¿En qué se parece un archivo secuencial a una lista enlazada?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. En Estados Unidos, los registros de ADN de todos los presos federales están ahora almacenados en una base de datos para utilizarlos en investigaciones criminales. ¿Sería ético permitir que se usara esta información para otros propósitos, por ejemplo, para investigaciones médicas? En caso afirmativo, ¿para qué propósitos sí que sería ético? En caso negativo, ¿por qué no? ¿Cuáles son los pros y los contras en cada caso?
2. ¿Hasta qué punto debería una universidad estar autorizada a hacer pública la información acerca de sus estudiantes? ¿Qué pasa con sus nombres y direcciones? ¿Y qué pasa con la distribución de datos sin identificar a los estudiantes? ¿Es su respuesta coherente con la que ha proporcionado a la Cuestión 1?
3. ¿Qué restricciones son apropiadas en relación con la construcción de bases de datos con información personal? ¿Qué información tiene derecho el gobierno a poseer acerca de sus ciudadanos? ¿Qué información tiene derecho a tener una empresa de seguros sobre sus clientes? ¿Qué información tiene una empresa derecho a poseer en relación con sus empleados? ¿Deberían implementarse controles en todos estos casos? En caso afirmativo, ¿cuáles?
4. ¿Es apropiado que una empresa de tarjetas de crédito venda a las empresas de marketing información acerca de los hábitos de compra de sus clientes? ¿Es aceptable que una empresa de comercialización de coches deportivos venda su lista de correo a una revista de coches deportivos? ¿Es aceptable que el organismo encargado de gestionar los impuestos en Estados Unidos venda a los intermediarios de bolsa los nombres y direcciones de aquellos contribuyentes que han obtenido ganancias de capital significativas? Si no es capaz de responder a estas preguntas con un simple sí o no y sin matizaciones, ¿cuál cree que sería una política aceptable?
5. ¿Hasta qué grado es responsable el diseñador de una base de datos del modo en que se utilice la información contenida en dicha base de datos?
6. Suponga que una base de datos permite por error un acceso no autorizado a la información que contiene. Si alguien obtiene y utiliza esa información de forma maliciosa, ¿hasta qué punto comparten los diseñadores de la base de datos la responsabilidad por el mal uso de esa información? ¿Depende su respuesta del esfuerzo que el criminal haya necesitado para descubrir el fallo en el diseño de la base de datos y obtener la información no autorizada?
7. El uso cada vez mayor de técnicas de minería de datos plantea numerosas cuestiones relativas a la ética y a la intimidad. ¿Cree que nuestro derecho a la intimidad habrá sido vulnerado si la minería de datos revela ciertas

características acerca de la población global de nuestra ciudad? ¿Cree que el uso de las técnicas de minería de datos promueve prácticas empresariales adecuadas o prácticas fraudulentas? ¿Hasta qué grado es correcto obligar a los ciudadanos a participar en la elaboración de un censo, sabiendo que va a extraerse de esos datos más información que la que se está solicitando explícitamente en los cuestionarios individuales? ¿Proporciona la minería de datos a las empresas de marketing una ventaja injusta con respecto a las audiencias a las que se dirigen sus mensajes publicitarios, que no sospechan nada acerca de las técnicas que se utilizan? ¿Hasta qué punto es buena o mala la práctica de construir perfiles de clientes?

8. ¿Hasta qué punto debería permitirse a una persona o a una empresa recopilar y almacenar información acerca de las personas? ¿Qué pasa si esa información que se recopila ya está disponible públicamente, aunque esté dispersa entre distintas fuentes? ¿Hasta qué grado deben las personas o empresas que recopilen esa información proteger la información obtenida?

9. Muchas bibliotecas ofrecen un servicio de referencias, para que los usuarios puedan solicitar la ayuda de un bibliotecario a la hora de buscar información. ¿Cree que la existencia de Internet y de las tecnologías de bases de datos ha hecho que este servicio quede obsoleto? En caso afirmativo, ¿cree que es un paso hacia adelante o un paso hacia atrás? En caso negativo, ¿por qué no? ¿Cómo afecta a la existencia de las bibliotecas la generalización de Internet y de la tecnología de bases de datos?

10. ¿Hasta qué grado cree que está usted mismo expuesto a la posibilidad de robo de su identidad? ¿Qué pasos puede dar para minimizar ese riesgo? ¿Qué daños cree que podría sufrir si fuera víctima de un robo de identidad? ¿Sería usted responsable en caso de que se produjera ese robo de identidad?

Lecturas adicionales

Beg, C. E. y T. Connolly. ***Database Systems: A Practical Approach to Design, Implementation and Management***, 4ª ed. Boston, MA: Addison-Wesley, 2005.

Berstein, A., M. Kifer y P. M. Lewis. ***Database Systems***, 2ª ed. Boston, MA: Addision-Wesley, 2006.

Date, C. J. ***An Introduction to Database Systems,*** 8ª ed. Boston, MA: Addison-Wesley, 2004.

Date, C. J. ***Databases, Types and the Relational Model,*** 3ª ed. Boston, MA: Addison-Wesley, 2007.

Elmasri, R. y S. Navathe. ***Fundamentals of Database Systems***, 6ª ed. Boston, MA: Addison-Wesley, 2011.

Patrick, J. J. ***SQL Fundamentals***, 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Silberschatz, A., H. Korth y S. Sudarshan. ***Database Systems Concepts***, 8ª ed. New York: McGraw-Hill, 2009.

Ullman, J. D. y J. D. Widom. ***A First Course in Database Systems***, 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2008.

CAPÍTULO

10

Gráficos por computadora

En este capítulo vamos a explorar el campo de los gráficos por computadora, un campo que está teniendo un impacto muy significativo en la producción de películas y videojuegos interactivos. De hecho, los avances en este campo están liberando a las empresas de producción de contenido audiovisual de las restricciones de la realidad y hay mucha gente que sostiene que las animaciones generadas por computadora pueden llegar pronto a eliminar la necesidad de utilizar actores tradicionales, estudios y medios fotográficos en los sectores de la televisión y del cine.

Las secciones marcadas con asterisco se sugieren como secciones opcionales.

Los gráficos por computadora son la rama de las Ciencias de la computación que aplica tecnología informática a la producción y manipulación de representaciones visuales. Se asocian con una amplia variedad de temas, incluyendo la presentación de textos, la generación de gráficos y diagramas, el desarrollo de interfaces gráficas de usuario, la manipulación de fotografías, la producción de videojuegos y la creación de películas animadas. Sin embargo, el término ***gráficos por computadora*** se utiliza cada vez más para hacer referencia a un campo específico denominado gráficos 3D y la mayor parte de este capítulo se concentrará en este tema. Comenzaremos definiendo lo que son los gráficos 3D y clarificando su papel dentro de esa interpretación más amplia del término gráficos por computadora.

10.1 El ámbito de los gráficos por computadora

Con la aparición de las cámaras digitales, se ha incrementado rápidamente la popularidad del software para la manipulación de imágenes codificadas digitalmente. Este software permite "retocar" fotografías, eliminando manchas y esos malditos "ojos rojos" que aparecen en los retratos, así como cortar y pegar partes de diferentes fotografías con el fin de crear imágenes que no reflejen necesariamente la realidad.

A menudo se aplican técnicas similares para crear efectos especiales en los sectores de la televisión y del cine. De hecho, dichas aplicaciones fueron uno de los motivos principales para que estos sectores pasaran de los sistemas analógicos, como por ejemplo las películas físicas, a las imágenes codificadas digitalmente. Entre las aplicaciones podemos citar el eliminar la aparición de los cables de soporte en las tomas, la superposición de múltiples imágenes o la producción de cortas secuencias de imágenes nuevas que se emplean para modificar la acción originalmente capturada por una cámara.

Además de software para manipular fotografías digitales e imágenes de películas, existe ahora una amplia variedad de paquetes software de utilidad/aplicación que sirven de ayuda para la producción de imágenes bidimensionales; estos paquetes software van desde los simples dibujos basados en líneas hasta sofisticados paquetes de creación artística (un ejemplo elemental muy conocido es esa aplicación de Microsoft denominada Paint). Como mínimo, estos programas permiten al usuario dibujar puntos y líneas, insertar formas geométricas simples como óvalos y rectángulos, rellenar de color regiones de la imagen y cortar y pegar partes específicas de un dibujo.

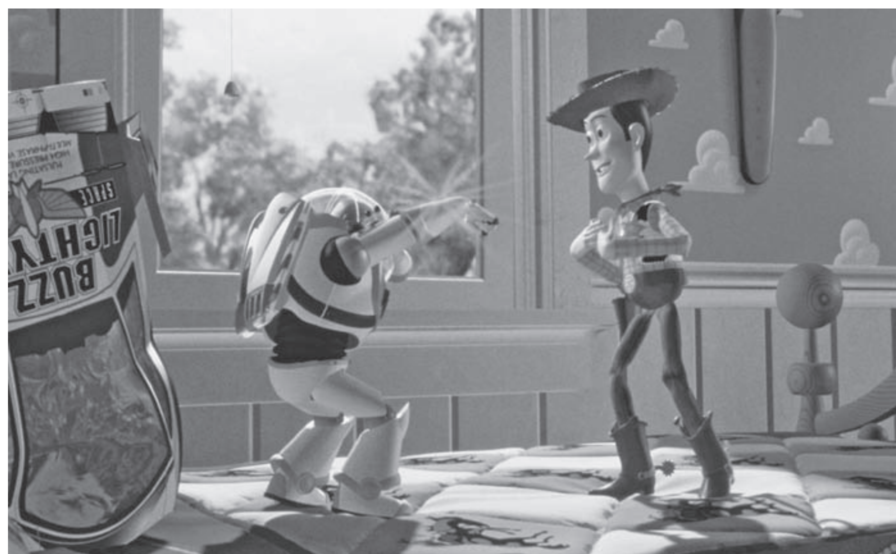
Observe que todas las aplicaciones anteriores tratan con la manipulación de formas e imágenes planas bidimensionales. Son, por tanto, ejemplos de dos campos relacionados de investigación: uno de ellos son los **gráficos 2D** y el otro es el **procesamiento de imágenes**. La distinción es que los gráficos 2D se centran en la tarea de convertir formas bidimensionales (círculos, rectángulos, letras, etc.) en patrones de píxeles con el fin de producir una imagen, mientras que el campo del procesamiento de imágenes, con el que nos toparemos de nuevo más adelante cuando abordemos el estudio de la inteligencia artificial, se centra en analizar los píxeles de una imagen con el fin de identificar patrones que puedan utilizarse para mejorar, o quizá, para "entender" la imagen. En resumen, los gráficos 2D tratan con la producción de imágenes mientras que el procesamiento de imágenes se ocupa del análisis de esas imágenes.

A diferencia del proceso de convertir formas bidimensionales en imágenes, como sucede en los gráficos 2D, el campo de los **gráficos 3D** trata con la conversión de formas tridimensionales en imágenes. El proceso consiste en construir versiones de escenas tridimensionales codificadas digitalmente y luego simular el proceso fotográfico, con el fin de generar imágenes de esas escenas. El objetivo es análogo al de la fotografía tradicional, salvo porque la escena que está siendo "fotografíada" mediante técnicas de gráficos 3D no existe como realidad física, sino que tan solo "existe" como un conjunto de datos y algoritmos. En otras palabras, los gráficos 3D implican "fotografiar" mundos virtuales (Figura 10.1), mientras que la fotografía tradicional pretende fotografiar el mundo real.

Es importante observar que la creación de una imagen mediante gráficos 3D implica dos pasos diferentes. Uno de ellos es la creación, codificación, almacenamiento y manipulación de la escena que se quiere fotografiar. El otro es el propio proceso de producción de la imagen. El primero es un proceso creativo, artístico; el segundo es un proceso computacionalmente muy costoso. Estos son temas que iremos viendo en las cuatro secciones siguientes.

El hecho de que los gráficos 3D generen "fotografías" de escenas virtuales los convierte en ideales para su uso en la producción de videojuegos interactivos y de películas, en donde las restricciones de la realidad limitarían la acción si no se emplearan estas técnicas avanzadas. Un videojuego interactivo está compuesto por un entorno virtual tridimensional codificado, con el que el jugador interactúa. Las imágenes que ve el jugador se generan mediante tecnología de gráficos 3D. Las películas animadas se crean de forma similar, salvo porque es el animador humano el que interactúa con el entorno virtual en lugar de que el que interactúe sea el espectador. El producto que al final se distribuye al

Figura 10.1 Una "fotografía" de un mundo virtual generada mediante gráficos 3D (de *Toy Story* por Walt Disney Pictures/Pixar Animation Studios) © Disney/Pixar.



público es una secuencia de imágenes bidimensionales, que estará determinada por el director/productor de la película.

En la Sección 10.6 investigaremos más en detalle el uso de los gráficos 3D en las animaciones. Por el momento, cerramos esta sección imaginando a dónde nos pueden conducir estas aplicaciones a medida que avance la tecnología de gráficos 3D. Hoy día, las películas se distribuyen como secuencias de imágenes bidimensionales. Aunque los proyectores que permiten visualizar esta información han progresado con respecto a los dispositivos analógicos que utilizaban cintas de películas, para terminar utilizando tecnología digital que emplea reproductores de DVD y pantallas planas, esos proyectores siguen teniendo que tratar únicamente con representaciones bidimensionales.

Imagine, sin embargo, cómo podría cambiar esto a medida que mejore nuestra capacidad de crear y manipular mundos virtuales tridimensionales realistas. En lugar de "fotografiar" estos mundos virtuales y distribuir una película en forma de secuencia de imágenes bidimensionales podríamos distribuir los mundos virtuales. Un espectador potencial recibiría acceso al "estudio" de la película, en lugar de limitarse a verla. Este estudio tridimensional sería entonces visualizado por medio de un "proyector de gráficos 3D", de forma bastante similar a como se visualizan los videojuegos utilizando "consolas de juegos" especializadas. El espectador podría primero contemplar una "trama sugerida", viendo así la película tal como la hubiera imaginado el director/productor. Pero el espectador podría también interactuar con el estudio virtual de una forma parecida a la de los videojuegos, con el fin de generar otros escenarios. Las posibilidades son inmensas, especialmente cuando consideramos también el potencial de las interfaces persona-máquina tridimensionales que se están desarrollando.

Cuestiones y ejercicios

1. Resuma la diferencia entre el procesamiento de imágenes, los gráficos 2D y los gráficos 3D.
2. ¿En qué sentido difieren los gráficos 3D de la fotografía tradicional?
3. ¿Cuáles son los dos pasos principales a la hora de producir una "fotografía" utilizando gráficos 3D?

10.2 Panorámica de los gráficos 3D

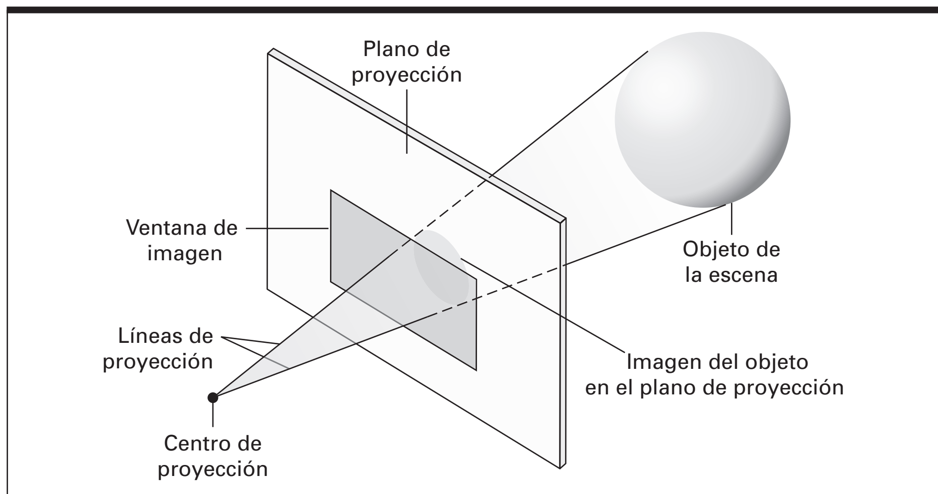
Iniciamos nuestro estudio sobre los gráficos 3D considerando el proceso completo de creación y visualización de imágenes, un proceso que consta de tres pasos: modelado, generación y visualización. La fase de modelado (que exploraremos en detalle en la Sección 10.3) es análoga al diseño y construcción de un estudio en la industria del cine tradicional, salvo porque la escena de gráficos 3D se "construye" a partir de datos y de algoritmos codificados digitalmente. Es decir, la escena generada en un contexto de utilización de gráficos por computadora puede que nunca llegue a existir en realidad.

El siguiente paso es generar una imagen bidimensional de la escena, calculando cómo aparecerían los objetos de la escena en una fotografía hecha con una cámara que estuviera situada en una posición específica. Este paso se denomina **generación** (*rendering*), que será el tema de las Secciones 10.4 y 10.5. La generación o reproducción implica aplicar las matemáticas de la geometría analítica para calcular la proyección de los objetos de la escena sobre una superficie plana, que se conoce con el nombre de **plano de proyección**; este proceso de proyección es en cierto modo análogo a la manera en la que una cámara proyecta una escena sobre una película (Figura 10.2). El tipo de proyección aplicada es una **proyección en perspectiva**, lo que quiere decir que todos los objetos se proyectan a lo largo de líneas rectas, denominadas **proyectores**, que salen de un punto común, denominado **centro de proyección** o **punto de vista**. (Este tipo de proyección difiere de la **proyección paralela** en la que las líneas de proyección son paralelas. Una proyección en perspectiva produce una imagen similar a la que normalmente percibe el ojo humano, mientras que una proyección paralela produce un "verdadero" perfil de un objeto, que a veces resulta útil en el contexto de los dibujos utilizados en el campo de la ingeniería.)

La parte restringida del plano de proyección que define las fronteras de la imagen final se conoce como **ventana de imagen**. Corresponde con el rectángulo que se muestra en la pantalla de la mayoría de las cámaras digitales para indicar los límites de la fotografía que se va a tomar. De hecho, la pantalla de la mayor parte de las cámaras permite ver una parte del plano de proyección de la cámara más grande que la propia ventana de imagen. (Puede que se vea en la pantalla la parte superior de la cabeza de la "tía Marta", pero a menos que toda la cabeza caiga dentro de la ventana de imagen, la pobre tía Marta aparecerá recortada en la fotografía final.)

Una vez identificada la parte de la escena que se proyecta sobre la ventana de imagen, habrá que calcular la apariencia de cada píxel de la imagen final. Este proceso píxel a píxel puede ser computacionalmente muy complejo, por-

Figura 10.2 El paradigma de los gráficos 3D.



que requiere determinar cómo interactúan los objetos de la escena con la luz: una superficie dura y brillante bajo una luz intensa tiene que mostrarse de forma diferente que una superficie suave y transparente bajo una luz indirecta. A su vez, el proceso de generación utiliza numerosas técnicas de distintos campos, incluyendo la ciencia de los materiales y la Física. Además, determinar la apariencia de un objeto requiere a menudo disponer de información de los otros objetos que componen la escena. El objeto podría estar dentro de la sombra proyectada por otro objeto, o bien el objeto puede ser un espejo, cuya apariencia será esencialmente la de algún otro objeto.

A medida que se determina la apariencia de cada píxel, los resultados se almacenan colectivamente en forma de representación de mapa de bits de la imagen en un área de almacenamiento denominada **memoria de fotograma**. Este buffer puede ser un área de la memoria principal o, en el caso del hardware diseñado específicamente para aplicaciones gráficas, puede ser un bloque de circuitos de memoria de propósito especial.

Finalmente, la imagen almacenada en la memoria de fotograma se visualiza directamente o se transfiere a un medio de almacenamiento más permanente para su visualización posterior. Si la imagen está siendo producida para su uso en una película, se la puede almacenar, e incluso también modificar, antes de su presentación final. Sin embargo, en un videojuego interactivo o en un simulador de vuelo las imágenes deben ser visualizadas a medida que son producidas en tiempo real, lo que constituye un requisito que a menudo limita la calidad de las imágenes creadas. Esta es la razón por la que la calidad gráfica de las producciones animadas distribuidas por las productoras de cine es superior a la de los videojuegos interactivos actuales.

Vamos a terminar nuestra introducción a los gráficos 3D analizando un sistema típico de videojuegos. El propio juego es, esencialmente, un mundo virtual codificado, junto con un software que permite que el jugador manipule dicho mundo. A medida que el jugador manipula el mundo virtual, el sistema genera repetidamente la escena y almacena las imágenes en la memoria de fotograma. Para satisfacer las restricciones de tiempo real, buena parte de este proceso de generación se lleva a cabo mediante hardware de propósito especial. De hecho, la presencia de este hardware es una de las características que diferencian a una consola de juegos de una computadora personal genérica. Por último, el dispositivo de visualización del sistema de juegos muestra el contenido de la memoria de fotograma, creando en el jugador la ilusión de que la escena va cambiando.

Cuestiones y ejercicios

1. Resuma los tres pasos implicados en la producción de una imagen utilizando gráficos 3D.
2. ¿Cuál es la diferencia entre el plano de proyección y la ventana de imagen?
3. ¿Qué es una memoria de fotograma?

10.3 Modelado

Un proyecto de gráficos 3D comienza de forma bastante similar a una función teatral: es preciso diseñar un "estudio" y hace falta recopilar o construir el utillaje adecuados. En la terminología de los gráficos por computadora, el estudio se denomina **escena** y el utillaje son los **objetos**. Recuerde que una escena de gráficos 3D es virtual porque está compuesta por objetos que se "construyen" como modelos codificados digitalmente, en lugar de ser estructuras físicas tangibles.

En esta sección vamos a explorar los temas relacionados con la "construcción" de objetos y escenas. Comenzaremos con los problemas del modelado de objetos individuales y concluiremos considerando la tarea de recopilar dichos objetos para formar una escena.

Modelado de objetos individuales

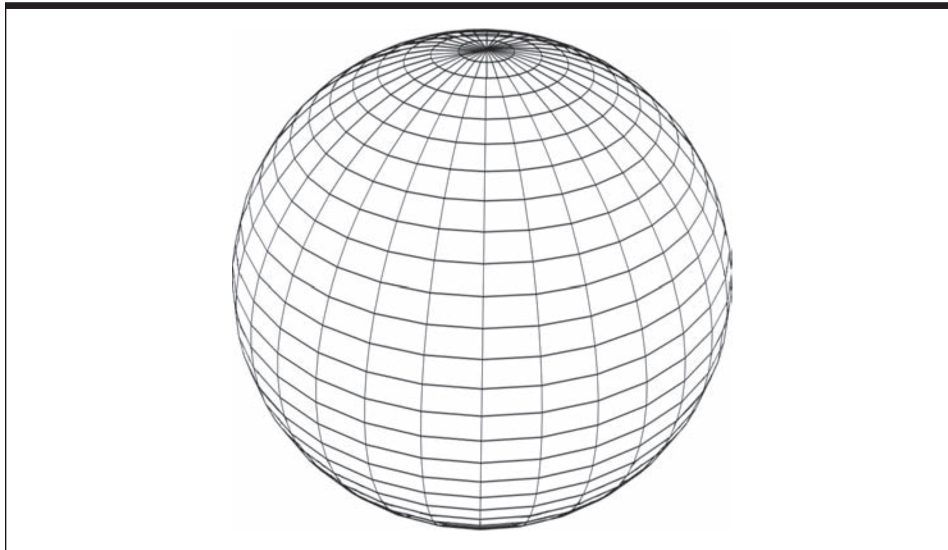
En la producción teatral, el grado con el que el escenario se adapta a la realidad depende de cómo se vaya a utilizar en la escena. Puede que no necesitemos poner en el escenario un automóvil completo, es posible que el teléfono no necesite funcionar y la escena de fondo puede estar pintada sobre una tela. De la misma forma, en el caso de los gráficos por computadora, el grado de precisión con el que el modelo software de un objeto tenga que reflejar las propiedades reales del objeto dependerá de los requisitos de cada situación concreta. Será necesario un mayor grado de detalle para modelar los objetos que vayan a estar en primer plano que los que vayan a ser objetos de fondo. Además, también puede generarse más detalle en aquellos casos que no estén sometidos a tremendas restricciones de tiempo real.

Algunos modelos de objetos pueden ser realmente simples, mientras que otros pueden tener una enorme complejidad. Como regla general, los modelos más precisos permiten obtener imágenes de mayor calidad, pero exigen tiempos más largos de generación. A su vez, buena parte de la investigación actual en el campo de los gráficos por computadora trata de desarrollar técnicas para la construcción de modelos de objetos altamente detallados, pero que sigan siendo eficientes. Parte de esta investigación se ocupa del desarrollo de modelos que puedan proporcionar diferentes niveles de detalle, dependiendo del papel último del objeto dentro de la escena; el resultado de ese desarrollo sería un único modelo de objeto que podría utilizarse dentro de un entorno cambiante.

La información requerida para describir un objeto incluye la forma del objeto, además de otras propiedades adicionales, como las características superficiales que determinan cómo interactuará el objeto con la luz. Por ahora, vamos a considerar la tarea de modelado de la forma.

Forma La forma de un objeto en el campo de los gráficos 3D suele describirse como un conjunto de pequeñas superficies planas denominadas **parches planos**, cada una de las cuales tiene forma de polígono. El conjunto de todos estos polígonos forma una **malla poligonal** que aproxima la forma del objeto que se está describiendo (Figura 10.3). Utilizando parches planos pequeños, la aproximación puede hacerse tan precisa como se desee.

Figura 10.3 Una malla poligonal para una esfera.



Los parches planos de una malla poligonal suelen elegirse de forma triangular, porque cada triángulo puede representarse mediante sus tres vértices, que es el número mínimo de puntos requerido para identificar una superficie plana en el espacio tridimensional. En cualquier caso, las mallas poligonales se representan mediante el conjunto de los vértices de sus parches planos componentes.

La representación mediante malla poligonal de un objeto puede obtenerse de varias formas. Una de ellas consiste en comenzar con una descripción geométrica precisa del objeto deseado y luego utilizar dicha descripción para construir una malla poligonal. Por ejemplo, la geometría analítica nos dice que una esfera (centrada en el origen) de radio r queda descrita por la ecuación

$$r^2 = x^2 + y^2 + z^2$$

Basándose en esta fórmula, podemos calcular las ecuaciones para las líneas de latitud y longitud de la esfera, identificar los puntos en los que esas líneas intersectan y luego usar esos puntos como vértices de una malla poligonal. Pueden aplicarse otras técnicas similares a otras formas geométricas tradicionales y esta es la razón por la que los personajes en las animaciones por computadora más sencillas a menudo parecen estar compuestos a partir de estructuras tales como esferas, cilindros y conos.

Las formas más generales pueden describirse por medios analíticos más sofisticados. Uno de ellos está basado en el uso de las **curvas de Bezier** (denominadas así por Pierre Bezier, que desarrolló el concepto a principios de la década de 1970 cuando trabajaba como ingeniero para la empresa automovilística Renault), que permiten definir un segmento de línea curva en el espacio tridimensional utilizando únicamente unos cuantos puntos, denominados puntos de control (dos de los cuales representan los extremos del segmento de curva, mientras que los otros indican cómo está distorsionada esa curva). Por ejemplo, la Figura 10.4 muestra una curva definida mediante cuatro puntos de

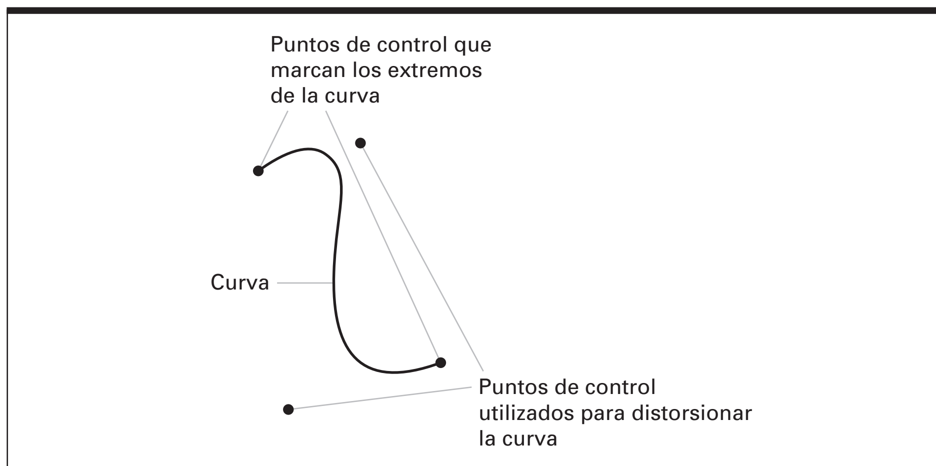
control. Observe cómo la curva parece haber sido estirada hacia los dos puntos de control que no identifican los extremos del segmento. Moviendo esos puntos de control, la curva puede doblarse para componer diferentes formas. (Es posible que el lector ya haya experimentado con este tipo de técnicas a la hora de construir líneas curvas con paquetes de dibujo como Paint de Microsoft.) Aunque no analizaremos el tema con más detalle aquí, las técnicas de Bezier para la descripción de curvas pueden ampliarse para describir superficies tridimensionales conocidas con el nombre de **superficies de Bezier**. A su vez, las superficies de Bezier han demostrado una gran eficiencia como primer paso en el proceso de obtención de mallas poligonales para superficies complejas.

Puede que se esté preguntando por qué es necesario convertir una descripción precisa de una forma geométrica, como por ejemplo la fórmula concisa de una esfera o las fórmulas que describen una superficie de Bezier, en una aproximación de esa forma geométrica utilizando una malla poligonal. La respuesta es que la representación de la forma de todos los objetos mediante mallas poligonales permite aplicar una solución uniforme al proceso de generación, un convenio que permite generar escenas completas de manera más eficiente. Así, aunque las fórmulas geométricas proporcionan descripciones precisas de las formas, se utilizan simplemente como herramientas para la construcción de mallas poligonales.

Otra forma de obtener una malla poligonal es construir la malla mediante la fuerza bruta. Esta técnica es popular en aquellos casos en los que una forma desafía todos los intentos de representarla mediante técnicas matemáticas elegantes. El procedimiento consiste en construir un modelo físico del objeto y luego anotar la ubicación de una serie de puntos de la superficie del modelo tocando la superficie con un dispositivo de tipo lápiz que registra su posición en el espacio tridimensional, un proceso que se conoce como **digitalización**. El conjunto de puntos obtenido se puede entonces utilizar como conjunto de vértices para la obtención de una malla poligonal que describa la forma del objeto.

Lamentablemente, algunas formas son tan complejas que obtener modelos realistas mediante un modelado geométrico o una digitalización manual no resulta factible. Entre los ejemplos podríamos citar las estructuras intrincadas

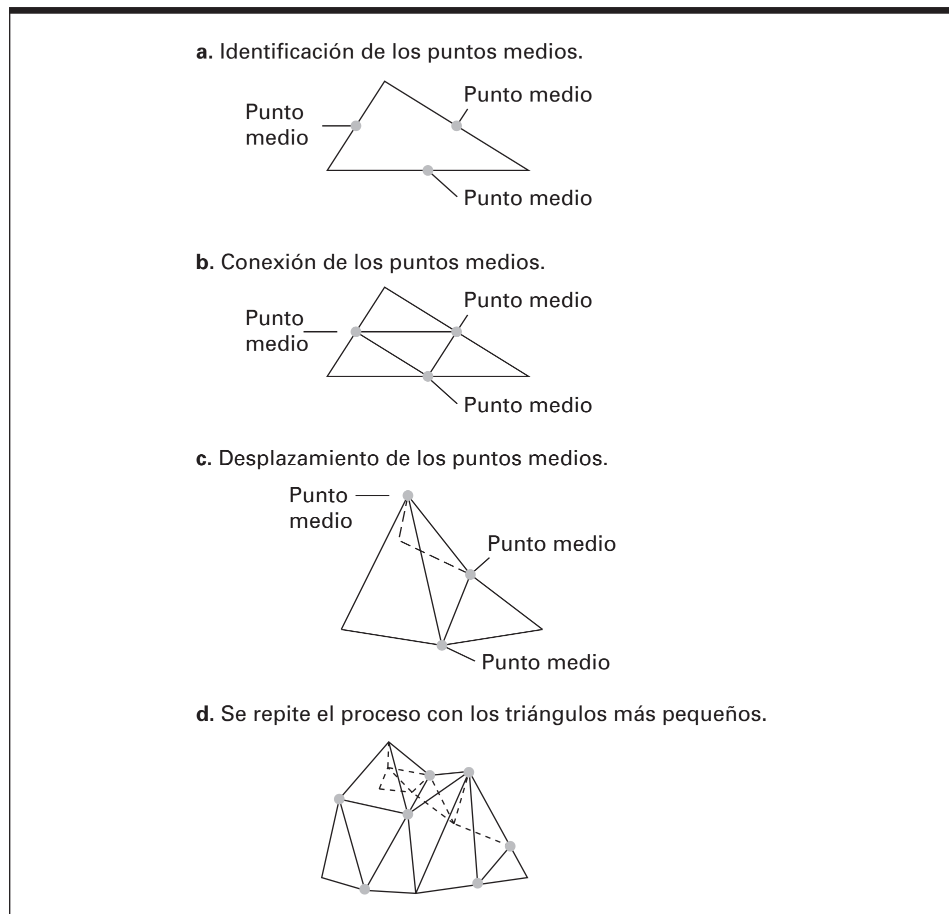
Figura 10.4 Curva de Bezier.



de algunas plantas, como los árboles; las secciones complejas de terrenos, como las cordilleras montañosas y sustancias gaseosas tales como nubes, el humo o las llamas de una hoguera. En estos casos, las mallas poligonales se obtienen escribiendo programas que generen automáticamente la forma geométrica deseada. Estos programas se conocen conjuntamente con el nombre de **modelos procedimentales**. En otras palabras, un modelo procedimental es un programa que aplica un algoritmo con el fin de generar una cierta estructura.

Por ejemplo, se han utilizado modelos procedimentales para generar cordilleras montañosas ejecutando los pasos siguientes: comenzando con un único triángulo, se identifican los puntos medios de las aristas del triángulo (Figura 10.5a). A continuación, se conectan estos puntos medios con el fin de formar un total de cuatro triángulos más pequeños (Figura 10.5b). Ahora, manteniendo fijos los vértices originales del triángulo, se desplazan los puntos intermedios en el espacio tridimensional (permitiendo que las aristas del triángulo se alarguen o contraigan) distorsionando así las formas triangulares (Figura 10.5c). Este proceso se repite con cada uno de los triángulos más pequeños (Figura 10.5d) y se sigue aplicando hasta obtener el grado de detalle deseado.

Figura 10.5 Creación de una malla poligonal para una cordillera montañosa.



Los modelos procedimentales proporcionan un medio eficiente de generar múltiples objetos complejos que sean similares entre sí, pero todos ellos distintos. Por ejemplo, un modelo procedimental se puede utilizar para construir una diversidad de objetos realistas que representen árboles, cada uno de ellos con su propia estructura de ramas, todas ellas similares. Una técnica para la obtención de esos modelos de árboles consiste en aplicar reglas de ramificación para "hacer crecer" esos objetos árbol de forma bastante similar a como un analizador sintáctico (Sección 6.4) construye un árbol de análisis sintáctico a partir de las reglas gramaticales. De hecho, el conjunto de reglas de ramificación utilizado en estos casos también se denomina gramática. Una gramática se puede diseñar para "hacer crecer" abetos, y emplear otra gramática distinta para "hacer crecer" olmos.

Otro método para construir modelos procedimentales consiste en simular la estructura subyacente de un objeto mediante un gran conjunto de partículas. Dichos sistemas se denominan **sistemas de partículas**. Usualmente, los sistemas de partículas aplican ciertas reglas predefinidas para mover las partículas

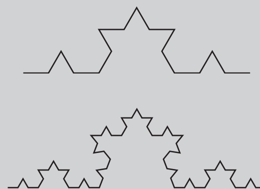
Fractales

La construcción de una cordillera montañosa por medio de un modelo procedimental, tal como se describe en el texto (véase la Figura 10.5) es un ejemplo del papel que desempeñan los fractales en los gráficos 3D. Técnicamente hablando, un **fractal** es un objeto geométrico cuya "dimensión de Hausdorff es mayor que su dimensión topológica". Lo que esto significa intuitivamente es que el objeto se forma "empaquetando" copias de un objeto de una dimensión menor. (Piense en la ilusión de anchura creada "empaquetando" múltiples segmentos de línea paralelos.) Un fractal normalmente se forma por medio de un proceso recursivo, en el que cada activación de la recursión "empaqueta" copias adicionales (aunque más pequeñas) del patrón que se está utilizando para construir el fractal. El fractal resultante es autosimilar, en el sentido de que cada porción, al ser ampliada, aparece como una copia del fractal completo.

Un ejemplo tradicional de fractal es el copo de nieve de von Koch, que se forma sustituyendo repetidamente los segmentos de línea recta de la estructura



por versiones más pequeñas de la misma estructura. Esto conduce a una secuencia de refinamientos que continúa de la forma siguiente:



Los fractales suelen ser la columna vertebral de los modelos procedimentales en el campo de los gráficos 3D. De hecho, han sido utilizados para generar imágenes realistas de cordilleras montañosas, de vegetación, de nubes y de humo.

que componen el sistema, quizá de una manera que recuerda en parte a las interacciones moleculares, con el fin de generar la forma geométrica deseada. Por ejemplo, se han utilizado sistemas de partículas para generar una animación de salpicaduras de agua, como veremos posteriormente al hablar de la animación. (Imagine un cubo de agua modelado como un cubo de canicas. A medida que el cubo se mueve, las canicas contenidas en el cubo también se mueven, simulando el movimiento del agua.) Otros ejemplos de aplicaciones de los sistemas de partículas serían escenas de llamas titilantes de un fuego, de nubes y de multitudes.

La salida de un modelo procedimental normalmente es una malla poligonal que se aproxima a la forma del objeto deseado. En algunos casos, como al generar una cordillera montañosa a partir de triángulos, la malla es una consecuencia natural del propio proceso de generación. En otros casos, como cuando se hace crecer un árbol mediante reglas de ramificación, la malla puede construirse como paso final adicional. Por ejemplo, en el caso de los sistemas de partículas, las partículas situadas en el borde exterior del sistema son candidatos naturales para actuar como vértices de la malla poligonal final.

El grado de precisión de la malla generada por un modelo procedimental puede depender de cada situación concreta. Un modelo procedimental para un árbol que vaya a aparecer al fondo de una escena podría generar una malla bastante burda que refleje solo la forma básica del árbol, mientras que un modelo procedimental para un árbol que vaya a aparecer en primer plano podría generar una malla en la que se distinguieran las ramas y las hojas individuales.

Características superficiales Un modelo compuesto simplemente de una malla poligonal captura únicamente la forma de un objeto. La mayoría de los sistemas de generación son capaces de enriquecer esos modelos durante el propio proceso de generación, con el fin de simular diversas características superficiales, a solicitud del usuario. Por ejemplo, aplicando diferentes técnicas de sombreado (que presentaremos en la Sección 10.4), un usuario puede especificar que la malla poligonal de una bola sea generada como una bola suave y de color rojo o como una bola rugosa de color verde. En algunos casos, esa flexibilidad es deseable, pero en aquellas situaciones en las que hace falta una generación fiel de un objeto original, es preciso incluir en el modelo información más específica acerca del objeto, para que el sistema de generación sepa qué es lo que tiene que hacer.

Existen diversas técnicas de codificar información acerca de un objeto más allá de su forma geométrica. Por ejemplo, junto con cada vértice de una malla poligonal, podría codificarse el color que el objeto original tiene en ese punto. Esta información podría entonces emplearse durante la generación para recrear la apariencia del objeto original.

En otros casos, pueden asociarse patrones de color con la superficie de un objeto por medio de un proceso conocido con el nombre de **aplicación de texturas**. La aplicación de texturas es similar al proceso de aplicar papel pintado a una pared, en el sentido de que asocia una imagen predefinida con la superficie del objeto. Esta imagen puede ser una fotografía digital, el dibujo realizado por un artista, o quizá una imagen generada por computadora. Imágenes tradicionalmente utilizadas como texturas son las de paredes de ladrillo, las superficies de madera veteada y las superficies de mármol

Suponga, por ejemplo, que deseáramos modelar una pared de ladrillos. Podríamos representar la forma de la pared con una simple malla poligonal compuesta por un rectángulo uniforme de gran longitud. Entonces podríamos suministrar junto con esta malla una imagen bidimensional de una pared de ladrillos. Después, durante el proceso de generación, esta imagen podría aplicarse al objeto rectangular para generar la apariencia de una pared de ladrillos. Para ser más precisos, cada vez que el proceso de generación necesita determinar la apariencia de un punto de la pared, simplemente utilizaría la apariencia del correspondiente punto en la imagen de la pared de ladrillos.

Cuando mejor funciona la aplicación de texturas es cuando se aplica a superficies relativamente planas. El resultado puede parecer artificial si hace falta distorsionar la imagen de la textura significativamente para cubrir una superficie curva (imagine los problemas que surgen al tratar de cubrir con papel pintado una pelota) o si la imagen de la textura envuelve completamente a un objeto, provocando que aparezca una junta allí donde el patrón de textura utilizado no encaje bien consigo mismo. En cualquier caso, la aplicación de texturas ha demostrado ser un medio eficiente de simular la textura de los objetos y se utiliza ampliamente en situaciones sensibles al tiempo real, como en el caso de los videojuegos interactivos.

Búsqueda de realismo La construcción de modelos de objetos que permitan obtener imágenes realistas es uno de los temas activos de investigación hoy día. De especial interés son los materiales asociados con personajes vivos, como por ejemplo la piel, el pelo y las plumas. Buena parte de estas investigaciones son específicas de cada sustancia concreta y abarcan técnicas tanto de modelado como de generación. Por ejemplo, para obtener modelos realistas de la piel humana, algunos investigadores han incorporado en los modelos el grado con el que la luz penetra en las capas de piel de la dermis y la epidermis, y tratan de modelar también el modo en el que los contenidos de dichas capas afectan a la apariencia de la piel.

Otro ejemplo es el relativo al modelado del cabello humano. Si el cabello se observa desde una cierta distancia, entonces puede que resulten suficientes las técnicas de modelado más tradicionales. Pero en los primeros planos puede resultar difícil conseguir un pelo de apariencia realista. Entre los problemas existentes están por ejemplo las propiedades translúcidas del cabello, la profundidad textural, la forma en que el cabello cae o se ensortija y la manera en que responde a fuerzas externas como el viento. Para resolver estos problemas, algunas aplicaciones recurren al procedimiento de modelar mechones individuales de cabello, que es una tarea ingente, porque el número de cabellos existente en la cabeza de una persona puede ser del orden de 100.000. Sin embargo, todavía más sorprendente es que algunos investigadores han construido modelos de cabello que tienen en cuenta la textura, la variación de color y las características mecánicas de cada pelo.

Otro ejemplo en el que se ha procurado conseguir un considerable grado de detalle es a la hora de modelar la ropa. En este caso, se han utilizado los detalles de los patrones de entretejido para conseguir diferencias texturales apropiadas entre los distintos tipos de tejido, como por ejemplo tejidos satinados; asimismo, se han combinado las propiedades detalladas de los distintos tipos de hilo con datos de patrones de tejido, con el fin de crear modelos de tejidos que

permiten obtener imágenes de primer plano extremadamente realistas. Además, los conocimientos de física y de ingeniería mecánica se han aplicado a las hebras individuales de materiales tejidos, con el fin de calcular imágenes que reflejan aspectos tales como el estiramiento de las hebras y las roturas del tejido.

La producción de imágenes realistas es un área activa de investigación que, como ya hemos dicho, incorpora técnicas tanto de los procesos de modelado como de los de generación. Normalmente, a medida que se van haciendo progresos, las nuevas técnicas se incorporan primero en aplicaciones que no están sujetas a restricciones de tiempo real, como por ejemplo el software gráfico de los estudios de producción de películas de cine, en las que existe un retardo significativo entre los procesos de modelado/generación y la presentación final de las imágenes. [Este tipo de avances puede observarse comparando atentamente los personajes de ***Toy Story 2*** de Disney (1999) con los de la película original ***Toy Story*** (1995). Por ejemplo, se aplicaron técnicas recientemente desarrolladas para mejorar el aspecto de las juntas entre las mallas poligonales que representan las características faciales, como por ejemplo los límites entre la nariz y el resto de la cara.] A medida que estas nuevas técnicas se refinan y se simplifican, terminan utilizándose en las aplicaciones de tiempo real, con lo que la calidad de los gráficos en estos entornos también mejora. Es muy posible que una verdadera interacción realista en tiempo real con distintos tipos de mundos virtuales no esté demasiado lejos en el futuro.

Modelado de escenas completas

Una vez que los objetos de una escena se han descrito adecuadamente y se han codificado digitalmente, a cada uno se le asigna una ubicación, un tamaño y una orientación dentro de la escena. Este conjunto de información se enlaza entonces con el fin de obtener una estructura de datos denominada **grafo de escena**. Además, el grafo de escena contiene enlaces a objetos especiales que representan a las fuentes de luz, así como a un objeto concreto que representa a la cámara. Es aquí donde se almacenan la ubicación, la orientación y las propiedades focales de la cámara.

Por tanto, un grafo de escena es análogo a la disposición de un estudio en la fotografía tradicional. Contiene la cámara, las luces, el utillaje y las imágenes de fondo (todo lo que vaya a contribuir a la apariencia final de la fotografía cuando se apriete el disparador), todo ello colocado en los lugares apropiados. La diferencia es que una composición fotográfica tradicional contiene objetos físicos, mientras que un grafo de escena contiene representaciones de objetos codificadas digitalmente. En resumen, el grafo de escena describe un mundo virtual.

Para recalcar el ámbito que abarca un grafo de escena, considere de nuevo la imagen de la Figura 10.1 e imagine el grafo de escena utilizado para producirla. Los personajes, la pared, la colcha de la cama, el cabecero, el paquete situado detrás de Buzz Lightyear (el soldado espacial), las molduras de la ventana, los árboles situados detrás de la ventana y las fuentes de luz fueron todos ellos modelados con el grado de detalle apropiado y representados en el grafo de escena. De hecho, algunos objetos que podríamos inicialmente considerar como una única estructura, como por ejemplo Woody (el muñeco cowboy),

estaban en realidad representados dentro del grafo de escena como conglomerados de numerosos objetos individuales.

El posicionamiento de la cámara dentro de una escena tiene muchas repercusiones. Como hemos dicho anteriormente, el detalle con el que se modelen los objetos dependerá de la ubicación de cada objeto dentro de la escena. Los objetos que aparecen en primer plano requieren un mayor grado de detalle que los objetos de fondo, y la diferencia entre lo que es el primer plano y lo que constituye el fondo depende de la posición de la cámara. Si la escena se va a utilizar en un contexto análogo al de un escenario teatral, entonces los papeles del primer plano y el fondo están bien establecidos y pueden construirse los modelos de los objetos correspondientemente. Sin embargo, si el contexto requiere que la posición de la cámara se altere para obtener diferentes imágenes, puede que sea necesario ajustar entre una "fotografía" y otra el grado de detalle proporcionado por cada modelo de objeto. Esta es una de las áreas de investigación actuales. Podríamos concebir una escena compuesta por modelos "inteligentes" que fueran refinando sus mallas poligonales y otras características a medida que la cámara se desplazara dentro de la escena.

Un ejemplo interesante de escenario con cámara móvil es el que se genera en los sistemas de realidad virtual en los que se permite a los seres humanos experimentar la sensación de moverse por un mundo imaginario tridimensional. El mundo imaginario está representado por un grafo de escena y las personas ven este entorno por medio de una cámara que se desplaza por la escena de acuerdo con los movimientos que la propia persona haga. En realidad, para proporcionar la percepción de profundidad necesaria para la visión en tres dimensiones se utilizan dos cámaras: una que representa el ojo derecho de la persona y otra que representa su ojo izquierdo. Mostrando la imagen obtenida por cada cámara delante del ojo apropiado, la persona tiene la ilusión de estar dentro de la escena tridimensional, y cuando se añaden sensaciones sonoras y táctiles a la experiencia, la ilusión puede ser bastante realista.

Televisión 3D

Existen varias tecnologías para producir imágenes 3D para televisión, pero todas dependen del mismo efecto visual estereoscópico, dos imágenes ligeramente distintas que lleguen a los ojos izquierdo y derecho serán interpretadas por el cerebro asignando a las imágenes una sensación de profundidad. Los mecanismos más baratos para conseguir este efecto requieren la utilización de gafas especiales con lentes de filtro. Las lentes coloreadas más antiguas (usadas en el cine en la década de 1950) o las más modernas lentes polarizadas filtran diferentes partes de una misma imagen proyectada en la pantalla, lo que da como resultado que a cada ojo llegue una imagen distinta. Otra tecnología más cara requiere el uso de gafas "activas" que obturan alternativamente las lentes izquierda y derecha de forma sincronizada con una televisión 3D que va conmutando rápidamente entre la imagen izquierda y la derecha. Finalmente, se están desarrollando televisiones 3D que no requieren gafas especiales ni emplear ningún tipo de casco. Utilizan complicadas matrices de filtros o lentes de ampliación en la superficie de la pantalla con el fin de proyectar las imágenes izquierda y derecha hacia la cabeza del espectador con ángulos ligeramente distintos, lo que hace que los ojos izquierdo y derecho contemplen imágenes diferentes.

Para terminar, debemos decir que la construcción de un grafo de escena es un paso fundamental dentro del proceso de generación de gráficos 3D. Puesto que el grafo contiene toda la información requerida para producir la imagen final, su terminación marca el final del proceso del modelado artístico y el comienzo del proceso computacionalmente muy costoso de generación de la imagen. De hecho, una vez construido el grafo de escena, la tarea gráfica pasa a ser la de calcular las proyecciones, determinar los detalles de las superficies en puntos específicos y simular los efectos de la luz, un conjunto de tareas que es en buena medida independiente de la aplicación concreta.

Cuestiones y ejercicios

1. A continuación se indican cuatro puntos (codificados utilizando el sistema de coordenadas rectangular tradicional) que representan los vértices de un parche plano. Describa la forma del parche. (Para aquellos que no hayan estudiado geometría analítica, cada tripla especifica cómo alcanzar el punto en cuestión partiendo de la esquina de una habitación. El primer número nos dice cuánto hay que caminar a lo largo de la pared situada a nuestra derecha. El segundo número indica cuánto hay que andar en una dirección paralela a la pared situada a nuestra izquierda. El tercer número nos dice cuánto tenemos que elevarnos respecto del suelo. Si un número es negativo, tendrá que imaginarse que es un fantasma y que puede caminar hacia atrás a través de las paredes y el suelo.)

   (0, 0, 0)   (0, 1, 1)   (0, 2, 1)   (0, 1, 0)

2. ¿Qué es un modelo procedimental?
3. Enumere algunos de los objetos que podrían estar presentes en un grafo de escena utilizado para generar la imagen de un parque.
4. ¿Por qué las formas se representan mediante mallas poligonales aún cuando podrían representarse de manera más precisa mediante ecuaciones geométricas?
5. ¿Qué es la aplicación de texturas?

10.4 Generación (*rendering*)

Ahora es el momento de analizar el proceso de generación, que implica determinar cómo aparecerán los objetos de un grafo de escena cuando se proyecten sobre el plano de proyección. Hay diversas formas de llevar a cabo la tarea de generación. Esta sección se centra en la técnica tradicional utilizada en la mayoría de los sistemas gráficos más populares (videojuegos, computadoras personales, etc.) del "mercado de consumo" actual. En las siguientes secciones investigaremos dos alternativas a esta técnica.

Comencemos con algo de información básica acerca de la interacción entre la luz y los objetos. Después de todo, la apariencia de un objeto está determinada por la luz emitida por ese objeto, así que determinar la apariencia de un

objeto equivale en último término a la tarea de simular el comportamiento de los rayos luminosos.

Interacción entre la luz y las superficies

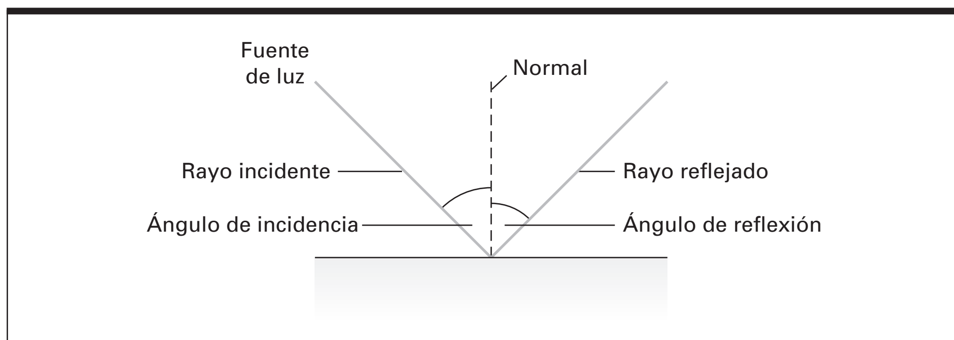
Dependiendo de las propiedades del material de un objeto, la luz que incide sobre su superficie puede ser absorbida, rebotar en la superficie en forma de luz reflejada o atravesar la superficie (y curvarse) como luz refractada.

Reflexión Consideremos un rayo de luz reflejado en una superficie opaca y plana. El rayo llega viajando en línea recta e incide sobre la superficie formando un ángulo denominado **ángulo de incidencia**. El ángulo con el que el rayo se refleja es idéntico al de incidencia. Como se muestra en la Figura 10.6, esos ángulos se miden respecto a una línea perpendicular (o **normal**) a la superficie (una línea normal a la superficie suele denominarse simplemente "normal" como por ejemplo en la frase "El ángulo de incidencia se mide respecto a la normal"). El rayo incidente, el rayo reflejado y la normal se encuentran en un mismo plano.

Si la superficie es suave, los rayos de luz paralelos (como por ejemplo los que lleguen de la misma fuente luminosa) que incidan sobre la superficie en una misma área se verán reflejados esencialmente en la misma dirección y saldrán del objeto en forma de rayos paralelos. Esa luz reflejada se denomina **luz especular**. Fíjese en que la luz especular solo puede observarse cuando la orientación de la superficie y de la fuente luminosa hace que la luz se refleje en dirección al espectador. Así, normalmente suele aparecer en forma de áreas brillantes sobre una superficie. Además, como la luz especular tiene un contacto mínimo con la superficie, tiende a ser del color de la fuente luminosa original.

Sin embargo, las superficies raramente son perfectamente suaves, por lo que muchos rayos luminosos pueden incidir sobre la superficie en determinados puntos cuya orientación difiere de la prevalente en esa superficie. Además, los rayos luminosos a menudo penetran en las capas superficiales y rebotan entre las partículas que componen la superficie, antes de salir finalmente en forma de luz reflejada. El resultado es que muchos rayos serán dispersados en distintas direcciones. Esta luz dispersada se denomina **luz difusa**. A diferencia de la luz especular, la luz difusa es visible en un amplio rango de direcciones.

Figura 10.6 Luz reflejada.

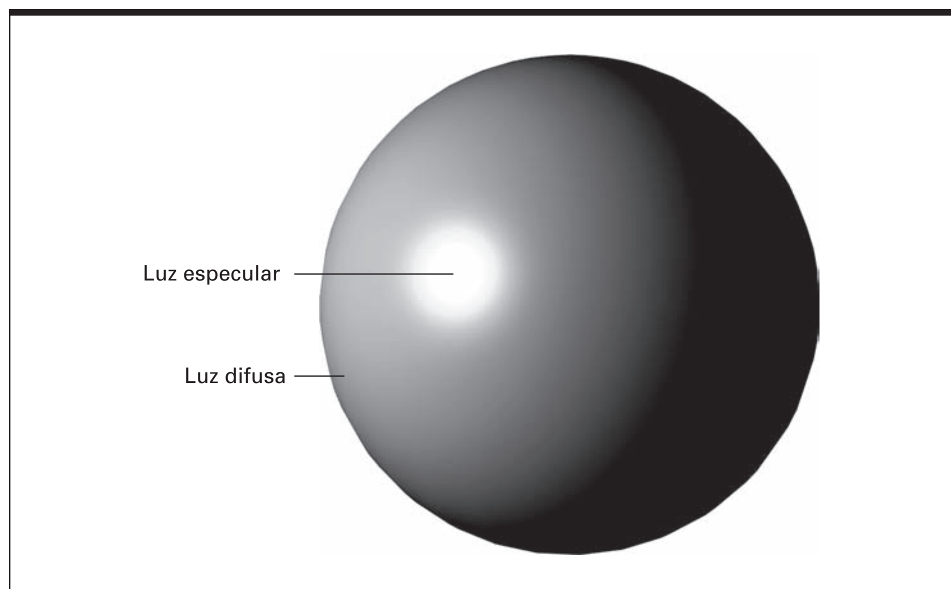


Puesto que tiende a tener un contacto prolongado con la superficie, la luz difusa es más susceptible a las propiedades de absorción del material y por tanto, tiende a adoptar el color del objeto.

La Figura 10.7 muestra una bola iluminada por una única fuente de luz. El área brillante en la superficie de la bola es producida por la luz especular. El resto del hemisferio orientado hacia la fuente de luz se percibe gracias a la luz difusa. Observe que el hemisferio orientado en dirección contraria a la fuente de luz principal no es visible gracias a la luz que se refleja directamente de esa fuente. La capacidad de ver esta otra parte de la bola se debe a la **luz ambiente**, que es luz dispersada que no está asociada con ninguna fuente luminosa ni ninguna dirección concreta. Las partes de las superficies iluminadas por la luz ambiente suelen aparecer con un color oscuro uniforme.

La mayor parte de las superficies reflejan tanto la luz especular como la difusa. Las características de la superficie determinan la proporción de cada tipo de luz. Las superficies suaves parecen brillantes porque reflejan más luz especular que difusa. Las superficies rugosas parecen más opacas porque reflejan más luz difusa que especular. Además, debido a las propiedades de detalle de algunas superficies, la relación entre luz especular y luz difusa varía dependiendo de la luz incidente. La luz que incida sobre ese tipo de superficies desde una dirección podría verse reflejada principalmente como luz especular, mientras que la luz que incida desde otra dirección podría verse reflejada principalmente como luz difusa. De ese modo, la apariencia de la superficie cambiaría de brillante a opaca a medida que fuera girando. Dichas superficies se denominan **superficies anisótropas**, por oposición a las **superficies isótropas**, cuyos patrones de reflexión son simétricos. Podemos encontrar ejemplos de superficies anisótropas en tejidos como el satín, en el que las características de la tela alteran la apariencia del material dependiendo de su orientación. En la superficie de hierba de un campo de atletismo, la disposición de la hierba (normal-

Figura 10.7 Luz especular y luz difusa.



mente determinada por la forma en la que se corta) produce efectos visuales anisótropos, como por ejemplo patrones de bandas claras y oscuras.

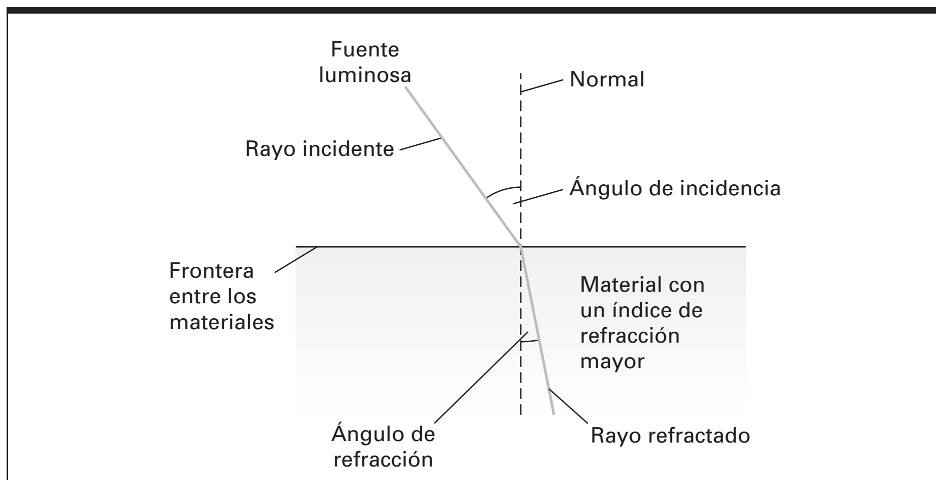
Refracción Consideremos ahora la luz que incide sobre un objeto que sea transparente en lugar de opaco. En este caso, los rayos luminosos atraviesan el objeto en lugar de rebotar en su superficie. A medida que los rayos penetran en la superficie su dirección se ve modificada, un fenómeno que se conoce como **refracción** (Figura 10.8). El grado de refracción está determinado por el índice de refracción de los materiales implicados. El índice de refracción está relacionado con la densidad del material. Los materiales densos tienden a tener un índice de refracción mayor que los materiales menos densos. Cuando un rayo de luz pasa a un material con un índice de refracción mayor (como por ejemplo al pasar del aire al agua), se curva hacia la normal en el punto de entrada. Si el rayo de luz pasa a un material con un índice de refracción menor, se curva alejándose de la normal.

Para generar objetos transparentes de forma apropiada, el software de generación debe conocer los índices de refracción de los materiales implicados. Pero la cosa no termina ahí. El software de generación también debe saber qué lado de la superficie de un objeto representa el interior del mismo, por oposición al exterior. ¿Está la luz entrando en el objeto o saliendo de él? Las técnicas para obtener este tipo de información son a veces bastante sutiles. Por ejemplo, si acordamos listar siempre los vértices de cada polígono de una malla poligonal en el sentido contrario a las agujas del reloj, visto el polígono desde fuera del objeto, entonces la lista codificada de manera inteligente indicará qué lado del parche representa la parte exterior del objeto.

Recorte, conversión de barrido y eliminación de caras ocultas

Ahora vamos a centrarnos en el proceso de producir una imagen a partir de un grafo de escena. Nuestro enfoque consistirá, por ahora, en seguir las técnicas utilizadas en la mayoría de los sistemas de videojuegos interactivos. Tomadas en conjunto, estas técnicas forman un paradigma muy bien consolidado que se

Figura 10.8 Luz refractada.

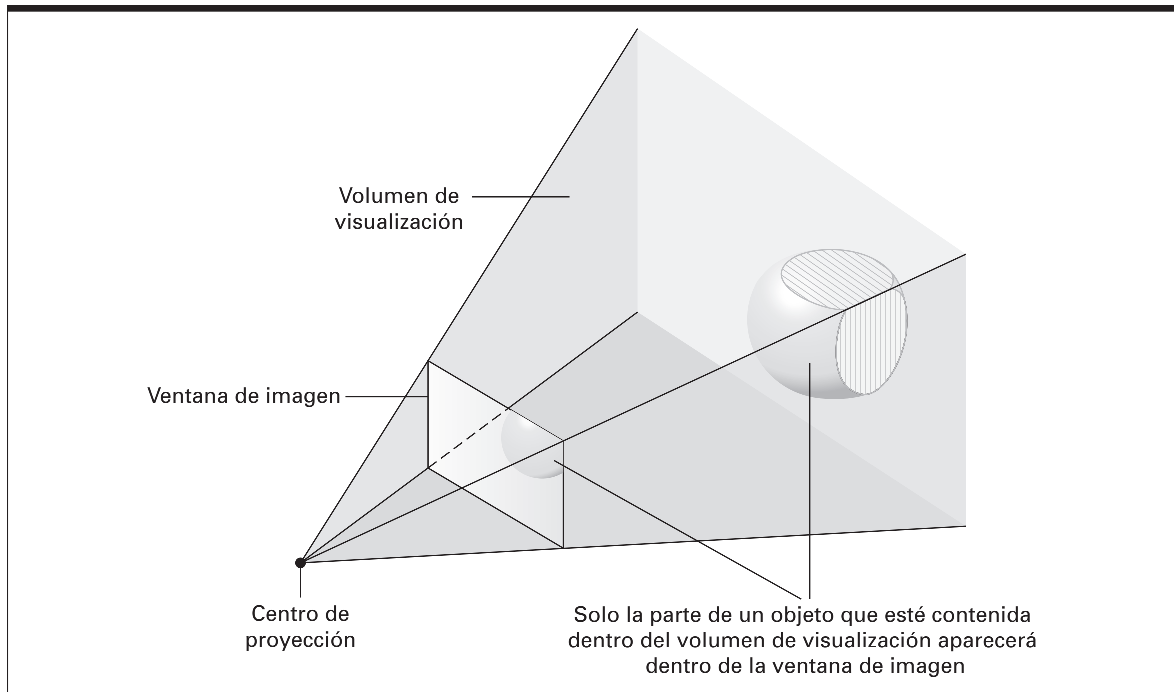


conoce con el nombre de **cadena de generación** (*rendering pipeline*). Consideraremos algunos de los pros y los contras de este enfoque al final de esta sección y en la siguiente sección exploraremos dos alternativas. Es útil observar que la cadena de generación maneja objetos opacos, por lo que la refracción no es un problema. Además, ignora las interacciones entre unos objetos y otros, por lo que por el momento no vamos a preocuparnos por elementos como los espejos y las sombras.

La cadena de generación comienza identificando la región dentro de una escena tridimensional que contiene los objetos (o partes de objetos) que pueden ser "vistos" por la cámara. Esta región, denominada **volumen de visualización**, es el espacio contenido dentro de la pirámide definida por las cuatro líneas rectas que salen del centro de proyección y pasan por las esquinas de la ventana de imagen (Figura 10.9).

Una vez identificado el volumen de visualización, la tarea consiste en descartar todos aquellos objetos o partes de objetos que no estén contenidos dentro del volumen de visualización. Después de todo, la proyección de esa parte de la escena caerá fuera de la ventana de imagen y, por tanto, no aparecerá en la imagen final. El primer paso consiste en descartar todos aquellos objetos que estén completamente fuera del volumen de visualización. Para simplificar este proceso, un grafo de escena puede estar organizado como una estructura de árbol en la que los objetos situados en distintas regiones de la escena estén almacenados en diferentes ramas. De esta manera pueden descartarse grandes secciones del grafo de escena simplemente ignorando determinadas ramas completas del árbol.

Figura 10.9 Identificación de la región de la escena que cae dentro del volumen de visualización.



Aliasing

¿Ha notado alguna vez la apariencia "parpadeante" que las corbatas y camisas de rayas tienen en las pantallas de televisión? Este es el resultado de un fenómeno conocido como **aliasing**, que se produce cuando un patrón de la imagen deseada se combina de manera inapropiada con la densidad de los píxeles que forman la imagen. Por ejemplo, suponga que una parte de la imagen deseada está compuesta por franjas alternativas blancas y negras, pero que el centro de todos los píxeles resulta caer únicamente en la franjas negras. En este caso, el objeto se representaría como completamente negro. Pero si el objeto se mueve ligeramente, el centro de todos los píxeles podría caer en la franjas blancas, lo que significa que el objeto cambiaría súbitamente pasando a ser de color blanco. Existen distintas formas de compensar este efecto tan molesto. Una de ellas consiste en generar cada píxel promediando una área pequeña de la imagen, en lugar de utilizar únicamente la apariencia de un único punto muy preciso.

Después de identificar y descartar los objetos que caen completamente fuera del volumen de visualización, los objetos restantes se recortan mediante un proceso denominado precisamente **recorte** (*clipping*), que esencialmente corta las partes de los objetos que caen fuera del volumen de visualización. Para ser más precisos, el recorte es el proceso de comparar cada parche plano individual con los límites del volumen de visualización y recortar aquellas partes del parche que quedan fuera. El resultado es una serie de mallas poligonales (de parches planos posiblemente recortados) que caen completamente dentro del volumen de visualización.

El siguiente paso de la cadena de generación consiste en identificar los puntos de los parches planos restantes que hay que asociar con cada posición de píxel en la imagen final. Es importante darse cuenta de que solo esos puntos contribuirán a la imagen final. Si un determinado detalle de un objeto cae entre dos posiciones de píxel no quedará representado por ningún píxel y no será visible por tanto en la imagen final. Esta es la razón por la que se indica claramente el número de píxeles que tiene cada aparato en el mercado de las cámaras digitales. Cuantos más píxeles haya, más probable es que podamos capturar en nuestras fotografías los pequeños detalles.

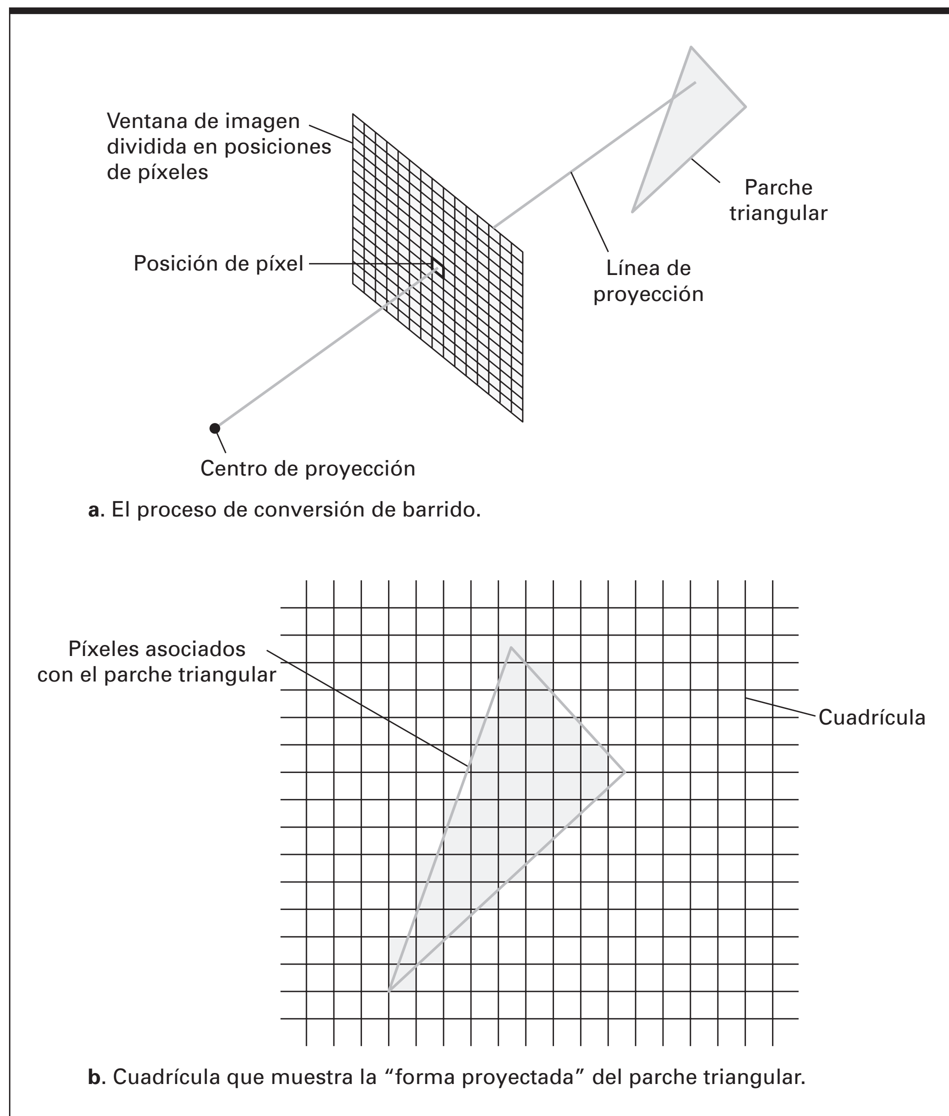
El proceso de asociar las posiciones de los píxeles con puntos de la escena se denomina **conversión de barrido** (porque implica convertir los parches en filas horizontales de píxeles, denominadas líneas de barrido) o **tramado** (*rasterization*, porque una matriz de píxeles es como una trama o "*raster*"). La conversión de barrido se lleva a cabo tirando líneas rectas (proyectores) desde el centro de proyección a través de cada posición de píxel de la ventana de imagen y luego identificando los puntos en los que estos proyectores intersectan con los parches planos. Estos serán entonces los puntos de los parches en los que deberemos determinar la apariencia de un objeto. De hecho, estos serán los puntos que se representarán mediante los píxeles en la imagen final.

La Figura 10.10 muestra la conversión de barrido de un único parche triangular. La parte (a) de la figura muestra cómo se utiliza un proyector para asociar una posición de píxel con un punto del parche. La parte (b) muestra la imagen en píxeles del parche, tal como ha sido determinada por la conversión

de barrido. Toda la matriz de píxeles está representada mediante una cuadrícula y en ella hemos sombreado los píxeles asociados con el triángulo. Observe que la figura también ilustra la distorsión que puede aparecer a la hora de realizar la conversión de barrido de una forma geométrica cuyas características sean pequeñas en relación con el tamaño de los píxeles. Esos bordes dentados les resultarán familiares a la mayoría de los usuarios de pantallas de computadoras personales.

Lamentablemente, el realizar la conversión de barrido de una escena completa (o incluso de un único objeto) no es tan sencillo como realizar la conversión de barrido de un único parche. Esto se debe a que, cuando hay múltiples parches implicados, uno de ellos puede bloquear la visión de otro. Así, aún cuando una línea de proyección intersecte con un parche plano, dicho punto

Figura 10.10 Conversión de barrido de un parche triangular.



del parche puede no ser visible en la imagen final. Identificar y descartar los puntos de una escena cuya visión está bloqueada es un proceso que se denomina **eliminación de caras ocultas**.

Una versión específica del proceso de eliminación de caras ocultas es lo que se denomina **eliminación de caras traseras**, que implica descartar todos los parches de una malla poligonal que representen la "parte trasera" de un objeto. Observe que la eliminación de caras traseras es relativamente sencilla, porque los parches del lado posterior de un objeto pueden identificarse fácilmente: son aquellos cuya orientación se aleja de la cámara.

Sin embargo, resolver el problema completo de eliminación de caras ocultas requiere mucho más que la simple eliminación de las caras traseras. Imagine por ejemplo una escena en la que hubiera un automóvil delante de un edificio. Habrá parches planos tanto del automóvil como del edificio que se proyectarán sobre la misma área de la ventana de imagen. Allí donde se produzca un solapamiento, los datos de los píxeles que terminen por almacenarse en la memoria de fotograma deberían indicar la apariencia del objeto situado en el primer plano (el automóvil) y no del objeto situado al fondo (el edificio). En resumen, si una línea de proyección intersecta con más de un parche plano, será el punto correspondiente al parche situado más cerca de la ventana de imagen el que habrá que generar.

Una solución simple para resolver este problema relativo a los objetos situados en el primer plano y el fondo es la que se conoce con el nombre de **algoritmo del pintor**, que consiste en disponer los objetos de la escena de acuerdo con su distancia a la cámara y luego realizar la conversión de barrido primero para los objetos más distantes, haciendo así que los resultados de la conversión de barrido de los objetos más próximos sustituyan a cualquier resultado que se haya obtenido previamente. Lamentablemente, el algoritmo del pintor falla en aquellos casos en los que hay objetos físicamente entremezclados. Por ejemplo, parte de un árbol podría encontrarse detrás de un objeto, mientras que otra parte del mismo árbol podría estar delante de ese objeto.

Otras soluciones más generales al problema de representar objetos de primer plano y de fondo se centran en los píxeles individuales en lugar de en objetos completos. Una técnica muy popular utiliza un área de almacenamiento adicional, denominada **buffer z** (y también buffer de profundidad), que contiene una entrada por cada píxel de la imagen (o, lo que es lo mismo, por cada entrada de píxel de la memoria de fotograma). Cada posición del buffer z se emplea para almacenar la distancia entre la cámara y el objeto actualmente representado por la correspondiente entrada de la memoria de fotograma; esa distancia se calcula a lo largo de la línea de proyección. Así, con la ayuda de un buffer z puede resolverse el problema de qué objetos están en primer plano y cuáles en el fondo, calculando y almacenando la apariencia de un píxel solo si los datos correspondientes a ese píxel no han sido almacenados todavía en la memoria de fotograma, o si el punto del objeto que estamos considerando actualmente está más próximo que el del objeto previamente generado; para ver si está más próximo se emplea la información de distancia almacenada en el buffer z.

Para ser más precisos, cuando se utiliza un buffer z, el proceso de generación puede llevarse a cabo de la forma siguiente: en todas las entradas del buffer z se almacena un valor que representa la distancia máxima con respecto a la cámara que puede tener un objeto, para que lo tengamos en consideración

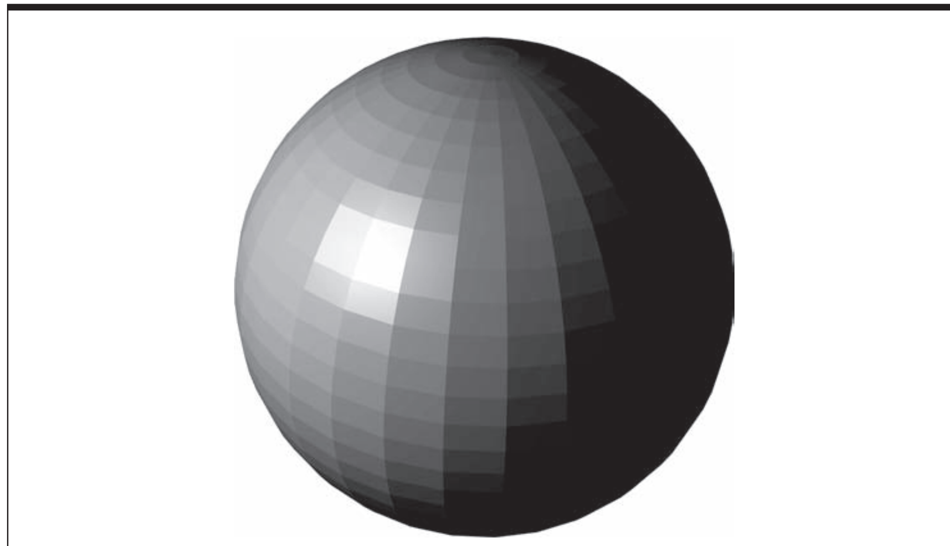
durante la generación. Después, cada vez que se tiene en cuenta un nuevo punto de un parche plano para generarlo, primero comparamos su distancia a la cámara con el valor del buffer z asociado con la posición del píxel actual. Si dicha distancia es inferior al valor almacenado en el buffer z, se calcula la apariencia del punto, se almacena el resultado en la memoria de fotograma y se sustituye la antigua entrada en el buffer z por la distancia al punto que se acaba de generar (observe que si la distancia al punto es mayor que el valor almacenado en el buffer z, no es necesario llevar a cabo ninguna acción, porque el punto de ese parche está demasiado alejado como para tenerlo en cuenta o su visión está bloqueada por otro punto más próximo que ya ha sido generado).

Sombreado (*shading*)

Una vez que el proceso de conversión de barrido ha identificado un punto de un parche plano que debe aparecer en la imagen final, la tarea de generación se convierte en la de determinar la apariencia del parche en dicho punto. Este proceso se denomina **sombreado**. Observe que el sombreado implica calcular las características de la luz proyectada hacia la cámara desde el punto en cuestión, características que dependerán a su vez de la orientación que la superficie tenga en dicho punto. Después de todo, es la orientación de la superficie en dicho punto la que determina el grado de luz especular, difusa y ambiente que será percibida por la cámara.

Una solución sencilla al proceso de sombreado, denominada **sombreado plano**, consiste en utilizar la orientación de un parche plano como la orientación de cada punto del parche; es decir, asumir que la superficie en cada parche es plana. Sin embargo, el resultado es que la imagen final aparecerá con facetas, como se muestra en la Figura 10.11 en lugar de redondeada como en la Figura 10.7. En un cierto sentido, el sombreado plano produce una imagen de la propia malla poligonal, en lugar de producir una imagen del objeto que está siendo modelado mediante dicha malla.

Figura 10.11 Una esfera tal como aparecería si se la genera con sombreado plano.

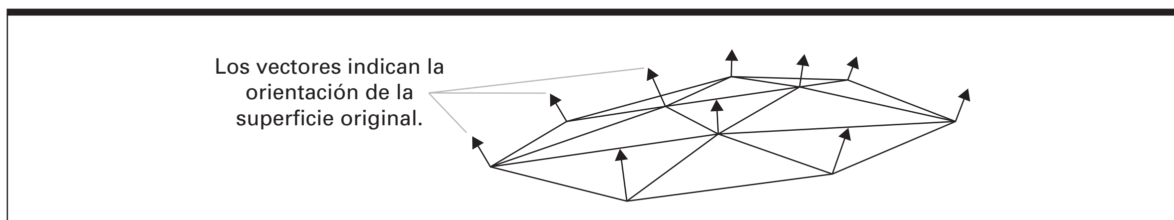


Para obtener una imagen más realista, el proceso de generación debe combinar la apariencia de los parches planos individuales para generar una superficie de aspecto suavemente curvo. Esto se consigue estimando la verdadera orientación de la superficie original en cada punto individual que se esté generando.

Dichos esquemas de estimación realizan normalmente su trabajo a partir de datos que indican la orientación de la superficie en los vértices de la malla poligonal. Hay diversas formas de obtener estos datos. Una de ellas es codificar la orientación de la superficie original en cada vértice y asociar estos datos a la malla poligonal como parte del proceso de modelado. Esto produce una malla poligonal con flechas, que se denominan **vectores normales**, asociadas a cada vértice. Cada vector normal apunta hacia fuera en la dirección perpendicular a la superficie original. El resultado es una malla poligonal que puede visualizarse como se muestra en la Figura 10.12 (otra técnica consiste en calcular la orientación de cada uno de los parches adyacentes a un vértice y luego utilizar una "media" de dichas orientaciones como estimación de la orientación de la superficie en el vértice).

Independientemente de cómo se obtenga la orientación de la superficie original en los vértices de una malla poligonal, tenemos a nuestra disposición diversas estrategias para el sombreado de un parche plano a partir de esos datos. Entre esas estrategias se incluyen el **sombreado de Gouraud** y el **sombreado de Phong**, siendo la distinción entre estos dos tipos bastante sutil. Ambos comienzan utilizando la información acerca de la orientación de la superficie en los vértices de un parche, con el fin de aproximar la orientación de la superficie a lo largo de las aristas del parche. Después, el sombreado de Gouraud aplica dicha información para determinar la apariencia de la superficie a lo largo de las aristas del parche y, finalmente, interpola esa apariencia de los bordes con el fin de estimar la apariencia de la superficie en los puntos interiores del parche. Por el contrario, el sombreado de Phong interpola la orientación de la superficie a lo largo de las aristas del parche con el fin de estimar la orientación de la superficie en los puntos situados en su interior y solo entonces pasa a considerar las cuestiones acerca de la apariencia de los puntos. En resumen, el sombreado de Gouraud convierte la información de orientación en información de color y luego interpola esa información de color. El sombreado de Phong interpola la información de orientación hasta estimar la orientación del punto en cuestión y luego convierte dicha información de orientación en información de color. El resultado es que el sombreado de Phong es más proclive a mostrar la luz especular en el interior de un parche porque es más sensible a los cambios en la orientación de la superficie. (Véase la Cuestión 3 al final de esta sección.)

Figura 10.12 Una vista conceptual de una malla poligonal con vectores normales en los vértices.



Finalmente, hay que decir que las técnicas básicas de sombreado pueden ampliarse con el fin de dar a una superficie una apariencia de textura. Un ejemplo denominado **aplicación de protuberancias** (*bump mapping*) es básicamente una forma de generar pequeñas variaciones en la orientación aparente de una superficie para que parezca ser rugosa. Es decir, la aplicación de protuberancias añade un cierto grado de aleatoriedad al proceso de interpolación aplicado por los algoritmos de sombreado tradicionales, de modo que la superficie global parece tener una textura como la ilustrada en la Figura 10.13.

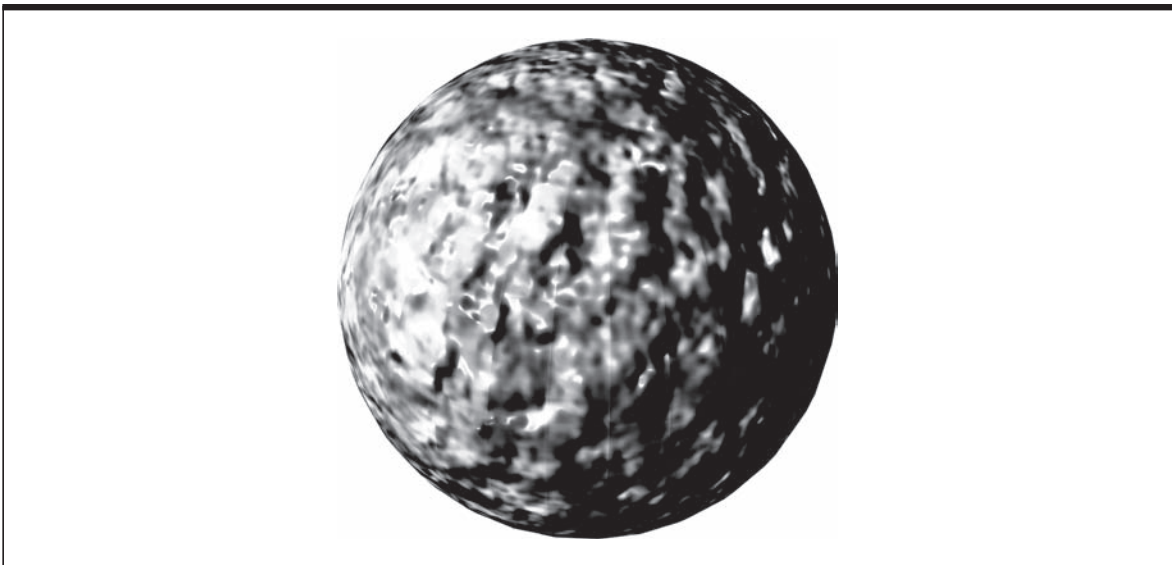
Hardware para la cadena de generación

Como ya hemos dicho, los procesos de recorte, conversión de barrido, eliminación de caras ocultas y sombreado forman una secuencia que se conoce con el nombre de cadena de generación. Además, los algoritmos para realizar estas tareas con eficiencia son bien conocidos y han sido implementados directamente en circuitos electrónicos, que se han miniaturizado mediante la tecnología VLSI con el fin de producir chips que realicen automáticamente toda la cadena de generación. Hoy día, incluso los dispositivos más baratos son capaces de generar millones de parches planos por segundo.

La mayoría de los sistemas de computadoras diseñados para aplicaciones de gráficos, incluyendo las consolas para videojuegos, incorporan estos dispositivos en su diseño. En el caso de sistemas de computadoras de propósito más general, esta tecnología puede añadirse en forma de **tarjeta gráfica** o **adaptador gráfico**, que se conecta al bus de la computadora como una controladora especializada (véase el Capítulo 2). Ese hardware reduce significativamente el tiempo requerido para ejecutar el proceso de generación.

El hardware de la cadena de generación también reduce la complejidad del software de aplicación gráfica. Básicamente, lo único que el software tiene que hacer es proporcionar el grafo de la escena al hardware de gráficos. El hard-

Figura 10.13 Una esfera tal como aparecería al ser generada utilizando una aplicación de protuberancias.



ware realiza entonces los pasos de la cadena y coloca los resultados en la memoria de fotograma. Así, desde la perspectiva del software, toda la cadena de generación se reduce a un único paso que utiliza al hardware como herramienta abstracta.

Por ejemplo, consideremos de nuevo el caso de un videojuego interactivo. Para inicializar el juego, el software transfiere el grafo de escena al hardware gráfico. El hardware entonces genera la escena y coloca la imagen en la memoria de fotograma, desde donde automáticamente se muestra en la pantalla del monitor. A medida que va progresando el juego, el software simplemente actualiza el grafo de escena con el que trabaja el hardware gráfico con el fin de reflejar los cambios en la situación del juego. Y el hardware genera repetidamente la escena, colocando cada vez la imagen actualizada en la memoria de fotograma.

Debemos observar, sin embargo, que la funcionalidad y las capacidades de comunicación de los distintos tipos de hardware gráfico existentes varían sustancialmente. Así, si se desarrolla una aplicación, como por ejemplo un videojuego para una plataforma específica, será necesario modificar esa aplicación si se transfiere a otro entorno. Para reducir esta dependencia con respecto a los detalles específicos de los sistemas gráficos, se han desarrollado interfaces software estándar que desempeñan un papel de intermediario entre el hardware gráfico y el software de aplicación. Estas interfaces están compuestas por rutinas software que convierten una serie de comandos estandarizados en las instrucciones específicas requeridas por un determinado sistema de hardware gráfico. Como ejemplos podemos citar **OpenGL** (*Open Graphics Library*, Biblioteca abierta de gráficos), que es un sistema no propietario desarrollado por Silicon Graphics y ampliamente utilizado en el sector de los videojuegos, así como **Direct3D**, que fue desarrollado por Microsoft para su uso en entornos Microsoft Windows.

Para terminar, debemos observar que, a pesar de todas las ventajas asociadas con la cadena de generación, también existen algunas desventajas, la más significativa es el hecho de que la cadena implementa únicamente un **modelo de iluminación local**, lo que quiere decir que la cadena genera cada objeto independientemente de los objetos restantes. Es decir, con un modelo de iluminación local, cada objeto se genera con respecto a las fuentes luminosas como si fuera el único objeto de la escena. El resultado es que las interacciones luminosas entre los objetos, como por ejemplo las sombras y las reflexiones, no se verán capturadas en la escena. Por el contrario, en un **modelo de iluminación global** sí que se tienen en cuenta las interacciones entre objetos. Hablaremos en la siguiente sección de dos técnicas para la implementación de un modelo de iluminación global. Por ahora, limitémonos a decir que estas técnicas exceden las capacidades del procesamiento en tiempo real de las tecnologías actuales.

Sin embargo, esto no significa que los sistemas que utilizan hardware de cadena de generación no sean capaces de producir algunos efectos de iluminación global. De hecho, se han desarrollado algunas técnicas muy inteligentes para sobreponerse a algunas de las restricciones impuestas por un modelo de iluminación global. En particular, puede simularse la apariencia de las **sombras proyectadas**, que son sombras que se proyectan sobre el suelo, dentro del contexto de un modelo de iluminación local; el procedimiento consiste

en hacer una copia de la malla poligonal del objeto que está proyectando la sombra, aplastar esa malla duplicada hasta hacerla plana, colocarla sobre el suelo y asignarle un color oscuro. En otras palabras, la sombra se modela como si fuera otro objeto, que luego puede generarse mediante el hardware tradicional de cadena de generación para generar la ilusión de una sombra. Dichas técnicas son populares tanto en aplicaciones de "consumo" como los videojuegos interactivos, como en aplicaciones de "nivel profesional" como por ejemplo los simuladores de vuelo.

Cuestiones y ejercicios

1. Indique la diferencia entre la luz especular, la luz difusa y la luz ambiente.
2. Defina los términos ***recorte*** y ***conversión de barrido***.
3. El sombreado de Gouraud y el sombreado de Phong pueden resumirse de la forma siguiente: el sombreado de Gouraud utiliza la orientación de la superficie de un objeto a lo largo de las aristas de un parche con el fin de determinar la apariencia de la superficie en las aristas y luego interpola esas apariencias en el interior del parche para determinar la apariencia de cada punto concreto del interior. El sombreado de Phong interpola las orientaciones de las aristas con el fin de calcular las orientaciones de los puntos en el interior del parche y luego utiliza esas orientaciones para determinar la apariencia de cada punto concreto. ¿Cómo diferirá la apariencia de un objeto si utilizamos un tipo de sombreado u otro?
4. ¿Cuál es la importancia de la cadena de generación?
5. Describa cómo podrían simularse las reflexiones en un espejo utilizando un modelo de iluminación local.

10.5 Iluminación global de las escenas

Los investigadores están trabajando actualmente en dos alternativas a la cadena de generación. Ambas implementan un modelo de iluminación global y proporcionan así la posibilidad de acabar con las restricciones del modelo de iluminación local que son inherentes a la cadena de generación tradicionalmente utilizada. Una de estas alternativas es la técnica de trazado de rayos y la otra es la de radiosidad. Ambas son procesos muy meticulosos y que requieren una enorme cantidad de tiempo de computación, como vamos a ver a continuación.

Trazado de rayos

El **trazado de rayos** es esencialmente el proceso de seguir un rayo luminoso hacia atrás, con el fin de localizar su origen. El proceso comienza seleccionando uno de los píxeles que hay que generar, identificando la línea recta que pasa por ese píxel y por el centro de proyección y luego trazando el rayo luminoso que incide sobre la ventana de imagen según esa línea. Este proceso de

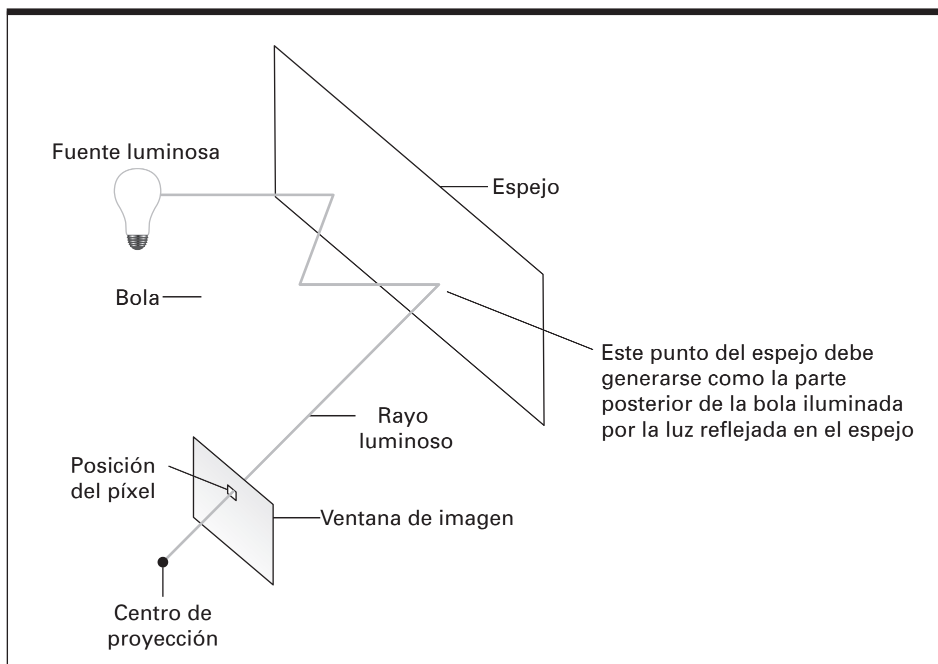
trazado implica seguir la línea hacia dentro de la escena hasta que entre en contacto con un objeto. Si el objeto es una fuente luminosa, el proceso de trazado de rayos termina y el píxel se genera como un punto de la fuente luminosa. En caso contrario, se evalúan las propiedades de la superficie del objeto para determinar la dirección del rayo luminoso incidente que fue reflejado, con el fin de producir el rayo que estamos siguiendo hacia atrás. El proceso sigue entonces hacia atrás a ese rayo incidente para localizar su origen, en cuyo punto puede que se identifique y que se tenga que trazar otro rayo.

En la Figura 10.14 se muestra un ejemplo de trazado de rayos en el que podemos ver cómo se traza hacia atrás un rayo a través de la ventana de imagen, hasta alcanzar la superficie de un espejo. A partir de ahí, el rayo se traza hasta una bola brillante, luego de nuevo hacia el espejo y de allí a la fuente luminosa. Basándose en la información obtenida durante este proceso de trazado, el píxel de la imagen debe aparecer como un punto de la bola, que está iluminado por la fuente luminosa reflejada en el espejo.

Una desventaja del trazado de rayos es que solo traza las reflexiones especulares. Por tanto, todos los objetos generados con este método tenderán a tener una apariencia brillante. Para contrarrestar este efecto, puede aplicarse una variante del trazado de rayos que se denomina **trazado de rayos distribuido**. La diferencia es que en lugar de trazar un único rayo hacia atrás a partir de un punto de reflexión, el trazado de rayos distribuido traza múltiples rayos a partir de dicho punto, cada uno de ellos apuntando en una dirección ligeramente distinta.

Otra variante del trazado de rayos básico es aplicable cuando la escena contiene objetos transparentes. En este caso, será preciso considerar dos efectos cada vez que se trace un rayo hacia atrás hasta alcanzar una superficie. Uno de

Figura 10.14 Trazado de rayos.



ellos es la reflexión y el otro es la refracción. Por ejemplo, observe la apariencia transparente del casco de Buzz (el guerrero espacial) en la Figura 10.1, así como los resaltes especulares próximos a la superficie superior del casco. En este caso, la tarea de trazar el rayo original se subdivide en dos tareas: trazar hacia atrás la reflexión y trazar también hacia atrás la refracción.

Normalmente, el trazado de rayos se implementa de forma recursiva, estando encargada cada activación de trazar un rayo hasta su fuente. La primera activación puede trazar su rayo hasta una superficie opaca brillante. En ese punto, reconocería que su rayo es la reflexión de un rayo incidente, calcularía la dirección de dicho rayo incidente e invocaría otra activación del proceso, con el fin de trazar ese rayo incidente. Esta segunda activación realizaría una tarea similar para encontrar el origen de su rayo, un proceso que podría resultar en que se invocaran otras activaciones adicionales.

Pueden utilizarse diversas condiciones para terminar con el trazado de rayos recursivo. El rayo puede trazarse hasta alcanzar una fuente luminosa, o bien el rayo puede salir de la escena sin incidir sobre ningún objeto o el número de activaciones puede alcanzar un límite predeterminado. Otra condición de terminación más puede estar basada en las propiedades de absorción de las superficies encontradas. Si una superficie es altamente absorbente, como por ejemplo una superficie oscura mate, entonces cualquier rayo incidente tendrá poco efecto sobre la apariencia de la superficie y el trazado de rayos puede darse por terminado. La absorción acumulativa también puede tener un efecto similar; es decir, el trazado de rayos puede terminar después de visitar varias superficies moderadamente absorbentes.

Estando basado en un modelo de iluminación global, el trazado de rayos está libre de muchas de las restricciones inherentes a la cadena de generación tradicional. Por ejemplo, los problemas de la eliminación de superficies ocultas y de detección de sombras están resueltas de forma natural en el proceso de trazado de rayos. Lamentablemente, el trazado de rayos tiene como principal desventaja el hecho de que requiere muchísimo tiempo de procesamiento. A medida que se traza cada reflexión hasta su fuente, el número requerido de cálculos crece enormemente, un problema que se complica todavía más cuando se permiten las refracciones o se aplica el trazado de datos distribuido. Por ello, el trazado de rayos no se implementa en los sistemas de tiempo real de "consumo", como los videojuegos interactivos, sino que donde podemos encontrarlo es en aplicaciones de tipo "profesional" que no son sensibles al tiempo real, como por ejemplo el software gráfico utilizado por los estudios de cine.

Radiosidad

Otra alternativa a la cadena de generación tradicional es la **radiosidad**. Mientras que el trazado de rayos va analizando punto a punto para trazar rayos luminosos individuales, la radiosidad adopta un enfoque más "regional" considerando la energía luminosa total radiada entre pares de parches planos. Esta energía luminosa radiada es esencialmente luz difusa. La energía luminosa radiada de un objeto es generada por ese objeto (como en el caso de una fuente luminosa) o reflejada en el objeto. La apariencia de cada objeto se determina entonces considerando la energía luminosa que recibe de otros objetos.

El grado con el que la luz radiada por un objeto afecta a la apariencia de otro está determinado por una serie de parámetros denominados **factores de forma**. Cada pareja de parches de la escena que hay que generar tiene asociado un factor de forma propio. Estos factores de forma tienen en cuenta las relaciones geométricas entre los parches indicados, como son distancia de separación y las orientaciones relativas. Para determinar la apariencia de un parche en la escena, se calcula la cantidad de energía luminosa recibida de todos los demás parches de la escena, utilizando el factor de forma apropiado en cada cálculo. Los resultados se combinan para generar unos valores únicos de color e intensidad para cada parche. Estos valores se interpolan después entre parches adyacentes, empleando técnicas similares al sombreado de Gouraud con el fin de obtener una superficie con contornos suaves, en lugar de una superficie con facetas.

Dado que son numerosos los parches que hay que considerar, la técnica de radiosidad es computacionalmente muy intensa. Por ello, como el trazado de rayos, su aplicación cae fuera de las capacidades de los sistemas gráficos de tiempo real actualmente disponibles en el mercado de consumo. Otro problema con la técnica de radiosidad es que, como trata con unidades compuestas por parches completos en lugar de con puntos individuales, no consigue capturar los detalles relativos a la luz especular, lo que significa que todas las superficies generadas mediante la técnica de radiosidad tienden a presentar una apariencia apagada.

Sin embargo, la radiosidad tiene sus ventajas. Una de ellas es que determinar la apariencia de los objetos utilizando la radiosidad es independiente de la cámara. Por tanto, una vez que se han realizado los cálculos de radiosidad para una escena, la generación de la escena puede completarse rápidamente para diversas posiciones de la cámara. Otra es que la radiosidad captura muchas de las características más sutiles de la luz, como el **sangrado de color**, que es el efecto por el que el color de un objeto afecta al tono de otros objetos situados cerca de él. Por ello, la radiosidad tiene sus propios nichos de mercado. Uno es el del software gráfico utilizado en entornos de diseño arquitectónico. De hecho, la luz existente dentro de un edificio que se esté diseñando está compuesta fundamentalmente por luz difusa y luz ambiente, de modo que los efectos especulares no son significativos, y el hecho de que se puedan gestionar de manera eficiente los cambios en la posición de la cámara implica que un arquitecto podrá ver rápidamente las distintas habitaciones desde distintas perspectivas.

Cuestiones y ejercicios

1. ¿Por qué el trazado de rayos sigue los rayos hacia atrás desde la ventana de imagen a la fuente luminosa, en lugar de ir hacia adelante desde la fuente luminosa a la ventana de imagen?
2. ¿Cuál es la diferencia entre el trazado de rayos simple y el trazado de rayos distribuido?
3. Cite dos desventajas de la técnica de radiosidad.
4. ¿En qué se parecen el trazado de rayos y la radiosidad? ¿En qué se diferencian?

10.6 Animación

Volvamos ahora nuestra atención hacia el tema de la animación por computadora, que es el uso de tecnología de computadoras para generar y visualizar imágenes en movimiento.

Fundamentos de la animación

Comenzamos presentando algunos conceptos básicos de animación.

Fotogramas La animación se consigue mostrando una secuencia de imágenes, denominadas **fotogramas**, en rápida sucesión. Estos fotogramas capturan la apariencia de una escena cambiante a intervalos de tiempo regulares, de modo que su presentación secuencial crea la ilusión de estar observando la escena continuamente a lo largo del tiempo. La velocidad estándar de visualización en la industria del cine es de 24 imágenes por segundo. El estándar en vídeo para televisión es de 60 o de 50 fotogramas por segundo (aunque como uno de cada dos fotogramas está diseñado para entremezclarse con el fotograma precedente, con el fin de generar una imagen detallada completa, el vídeo también puede clasificarse como un sistema de 30 o de 25 fotogramas por segundo).

Los fotogramas pueden producirse mediante técnicas fotográficas tradicionales o pueden generarse artificialmente por medio de sistemas de gráficos por computadora. Es más, las dos técnicas pueden combinarse. Por ejemplo, a menudo se utiliza software de gráficos 2D para modificar imágenes fotográficas, por ejemplo para eliminar los cables en una escena, superponer imágenes o crear la ilusión de **morfismo**, que es el proceso mediante el que un objeto parece transformarse en otro.

Un examen más detallado del proceso de morfismo proporciona algunas interesantes cuestiones acerca del proceso de animación. La construcción de un efecto de morfismo comienza identificando un par de fotogramas clave que serán los que enmarquen la secuencia de morfismo. Uno de ellos será la última imagen antes de que el morfismo tenga lugar. El otro será la primera imagen después de que ya haya ocurrido. (En la producción de películas tradicionales

Kineografías

Una kineografía es un libro de fotogramas de animación que simula el movimiento cuando se hacen pasar las páginas rápidamente. Cualquiera puede hacer una kineografía con este libro de texto (suponiendo que no haya llenado ya los márgenes de garabatos). Simplemente, dibuje un punto en el margen de la primera página y luego basándose en la posición de lo que hay impreso en la tercera página, coloque otro punto en esa tercera página en una posición ligeramente distinta al de la primera. Repita este proceso en cada página impar consecutiva hasta llegar al final del libro. Ahora, haga pasar rápidamente las páginas con el pulgar y mire cómo se mueve el punto de un lado a otro. ¡Ya está! Acaba de crear su propia kineografía y quizá haya dado así el primer paso en su carrera dentro del sector de la animación. Para experimentar con la Cinemática, trate de dibujar una figura con una serie de líneas en lugar de un simple punto, y haga que la figura parezca caminar. Ahora experimente con la Dinámica generando la imagen de una gota de agua al chocar con el suelo.

esto requiere "filmar" dos secuencias de la acción: una que conduce hasta el momento que comienza el morfismo y la otra que continúa después de que el morfismo se haya producido.) Determinadas características geométricas, como por ejemplo puntos y líneas, denominadas **puntos de control**, en el fotograma que precede al morfismo se asocian con otras características similares del fotograma posterior al morfismo y luego se construye el morfismo aplicando técnicas matemáticas que van distorsionando incrementalmente una imagen y aproximándola a la otra, usando como guías los puntos de control. Grabando las imágenes generadas durante este proceso de distorsión, se obtiene una corta de secuencia de imágenes producidas artificialmente que rellena el hueco existente entre los fotogramas clave originales y crea la ilusión del morfismo.

El guión Un proyecto típico de animación comienza con la creación de un **guión** (*storyboard*), que es una secuencia de imágenes bidimensionales que narran la historia completa en forma de bocetos de escenas correspondientes a puntos clave de la presentación. El papel que desempeñe el guión dependerá de si el proyecto de animación se implementa utilizando técnicas 2D o 3D. En un proyecto que emplee gráficos 2D, el guión suele evolucionar hasta convertirse en el conjunto final de fotogramas, de forma bastante similar a como lo hacía en los estudios Disney en la década de 1920. En aquellos días, una serie de artistas, llamados maestros animadores, iba refinando el guión para generar fotogramas detallados, denominados **fotogramas clave**, que establecían la apariencia de los personajes y de la escena a intervalos regulares de la animación. Después, los ayudantes de animación dibujaban fotogramas adicionales que permitían rellenar los huecos entre los fotogramas clave, de modo que la animación pudiera visualizarse de forma continua y suave. Este proceso de rellenado de huecos se denominaba **interpolación**.

La principal diferencia entre este proceso y el utilizado hoy día es que los animadores ahora utilizan software de gráficos 2D y de procesamiento de imágenes para dibujar los fotogramas clave, mientras que buena parte del proceso de interpolación se ha automatizado, con lo que el papel de los ayudantes de animación ha desaparecido en la práctica.

Animación 3D La mayor parte de las animaciones para videojuegos y producciones cinematográficas se crean ahora utilizando gráficos 3D. En estos casos, el proyecto sigue comenzando por la creación de un guión compuesto por imágenes bidimensionales. Sin embargo, en lugar de evolucionar hasta convertirse en el producto final, como en los proyectos de gráficos 2D, el guión se utiliza como guía en la construcción de un mundo virtual tridimensional. Este mundo virtual se "fotografía" después repetidamente, a medida que los objetos contenidos en él se mueven, de acuerdo con el guión o con la progresión del videojuego.

Quizá debiéramos hacer aquí una pausa para aclarar lo que queremos decir al afirmar que un objeto se mueve dentro de una escena generada por computadora. Recuerde que el "objeto" es en realidad un conjunto de datos almacenado en el grafo de la escena. Entre los datos de este conjunto, hay una serie de valores que indican la posición y la orientación del objeto. Por tanto, "mover" un objeto consiste simplemente en ir cambiando estos valores. Una vez realizados estos cambios, se utilizarán los nuevos valores durante el proceso de generación. En consecuencia, el objeto parecerá haberse movido en la imagen bidimensional final.

Desenfoque

En el campo de la fotografía tradicional, se han hecho muchos esfuerzos para tratar de generar imágenes nítidas de objetos que se mueven rápidamente. En el campo de la animación, el problema es justamente el opuesto. Si cada fotograma de una secuencia que muestra un objeto en movimiento presenta ese objeto como una imagen perfectamente definida, entonces el movimiento puede parecer "entrecortado". Sin embargo, las imágenes perfectamente nítidas son la consecuencia natural de la creación de fotogramas como imágenes individuales de objetos estáticos en un grafo de escena. Por ello, los animadores suelen distorsionar artificialmente las imágenes de los objetos en movimiento en un fotograma generado por computadora. Una de las técnicas utilizadas, denominada **supermuestreo**, consiste en producir múltiples imágenes en las que el objeto en movimiento está ligeramente desplazado de una a otra, y luego superponer dichas imágenes para obtener un único fotograma. Otra técnica consiste en alterar la forma del objeto en movimiento de manera que parezca alargarse en la dirección del movimiento.

Cinemática y Dinámica

El grado con el que el movimiento dentro de una escena de gráficos 3D está automatizado o controlado por un animador humano varía de una aplicación a otra. El objetivo, por supuesto, es automatizar el proceso completo. Con este fin, una gran parte de las investigaciones se han dirigido a tratar de encontrar formas de identificar y simular el movimiento de los fenómenos que tienen lugar de manera natural. A este respecto, hay dos áreas de la Mecánica que han resultado ser particularmente útiles.

Una de ellas es la **Dinámica**, que trata de describir el movimiento de un objeto aplicando las leyes de la Física con el fin de determinar el efecto de las fuerzas que actúan sobre el objeto. Por ejemplo, además de una posición, a un objeto de la escena se le puede asignar una dirección de movimiento, una velocidad y una masa. Estos elementos pueden entonces utilizarse para determinar el efecto que la gravedad o las colisiones con otros objetos tendrían sobre el objeto en cuestión, lo que permitiría al software calcular la posición correcta del objeto en el siguiente fotograma.

Por ejemplo, considere la tarea de construir una secuencia animada que muestre un chorro de agua cayendo en un contenedor. Podríamos utilizar un sistema de partículas en el grafo de escena para representar el agua, en el que cada partícula representaría una pequeña unidad de agua (piense en el agua como si estuviera compuesta de grandes "moléculas" del tamaño de canicas). Después, podríamos aplicar las leyes de la Física para calcular los efectos de la gravedad sobre las partículas, así como la interacción entre las propias partículas, por ejemplo si empezamos a mover el contenedor de lado a lado. Esto nos permitirá calcular la posición de cada partícula a intervalos de tiempo regulares y, utilizando la posición de las partículas exteriores como vértices de una malla poligonal, podríamos obtener una malla que describiera la superficie del agua. Nuestra animación podría entonces obtenerse "fotografiando" repetidamente esta malla, a medida que va progresando la simulación.

La otra rama de la Mecánica utilizada para simular el movimiento es la **Cinemática**, que trata de describir el movimiento de un objeto en términos de

cómo se mueven las partes del objeto unas con respecto a otras. Las aplicaciones de la Cinemática son muy importantes a la hora de animar figuras articuladas, en las que es necesario mover extremidades tales como brazos y piernas. Estos movimientos se pueden modelar más fácilmente simulando patrones de movimiento de las articulaciones, que calculando los efectos de las fuerzas individuales ejercidas por los músculos y la gravedad. Así, mientras que la Dinámica puede ser la técnica preferida a la hora de determinar la trayectoria de una pelota que rebota en el suelo, el movimiento del brazo de un personaje animado se determinaría aplicando la Cinemática para calcular las rotaciones apropiadas de las articulaciones del hombro, el codo y la muñeca. Por ello, muchas investigaciones en el campo de la animación de personajes vivientes se centran en los problemas de anatomía y en cómo la estructura de las articulaciones y las extremidades influyen en el movimiento.

Un método típico de aplicar la Cinemática consiste en comenzar representando un personaje mediante una figura compuesta por líneas rectas, que simula la estructura del esqueleto del personaje que se quiere animar. Cada sección de la figura se cubre luego con una malla poligonal que representa la superficie del personaje en las proximidades de dicha sección y se establecen una serie de reglas para determinar cómo hay que conectar entre sí las mallas adyacentes. Después, la figura puede manipularse (mediante software o un animador humano) reposicionando las articulaciones de la estructura del esqueleto, de la misma manera que se manipularía una marioneta. Los puntos en los que las "cuerdas" de la marioneta se conectan al modelo se denominan **avars**, que es la abreviatura de "*articulation variables*", variables de articulación (o más recientemente, "*animation variables*", variables de animación).

En la práctica, los *avars* se utilizan para controlar algo más que las simples posiciones de las articulaciones del esqueleto. Por ejemplo, el cowboy llamado Woody de *Toy Story* (Figura 10.1) tenía aproximadamente 100 avars asociados solo con la cara, lo que permitía a los animadores ajustar sus rasgos faciales con el fin de expresar emociones y mover la boca para que se ajustará a las palabras pronunciadas por el personaje.

Gran parte de la investigación en la aplicación de la Cinemática ha estado dirigida hacia el desarrollo de algoritmos para calcular automáticamente secuencias de posiciones de las extremidades que imiten el movimiento natural. Debido a ello, ahora hay disponibles algoritmos que generan secuencias realistas de movimientos tales como el caminar.

Sin embargo, gran parte de la animación basada en la Cinemática sigue generándose por el procedimiento de dirigir a un personaje a través de una secuencia predefinida de posiciones de las articulaciones y las extremidades. Estas posiciones pueden ser definidas gracias a la creatividad de un animador o pueden obtenerse mediante la técnica de **captura de movimiento**, que implica registrar las posiciones de un modelo vivo a medida que el modelo realiza la acción deseada. Para ser más precisos, después de aplicar cinta reflectante en determinados puntos estratégicos del cuerpo de una persona, puede fotografiarse a esa persona desde múltiples ángulos mientras que lanza una pelota. Después, observando la ubicación de la cinta reflectante en las diversas fotografías, se pueden identificar las orientaciones precisas de los brazos y las piernas de esa persona a medida que progresa la acción de lanzar la bola y esas

orientaciones pueden luego transferirse a un personaje como parte de una secuencia de animación.

El proceso de animación

El objetivo último de las investigaciones en el campo de la animación consiste en automatizar el proceso completo de animación. Podríamos imaginarnos un software que, a partir de los parámetros apropiados, generara automáticamente la secuencia de animación deseada. El progreso hecho en esta dirección queda ilustrado por el hecho de que la industria del cine ahora genera imágenes de multitudes, escena de batallas y estampidas de animales por medio de "robots" virtuales individuales que se mueven automáticamente dentro de un grafo de escena, realizando cada uno de ellos la tarea que tiene asignada.

Un caso interesante es el que tuvo lugar al filmar los ejércitos fantásticos de orcos y de humanos para la trilogía de ***El señor de los anillos***. Cada guerrero que aparecía en pantalla estaba modelado como un objeto "inteligente" diferente, con sus propias características físicas y con una personalidad asignada aleatoriamente, que le proporcionaba las tendencias de atacar o de huir. En las simulaciones de prueba para la batalla del Abismo de Helm en la segunda parte de la trilogía, los orcos tenían configurada su tendencia a huir con un valor demasiado alto, por lo que simplemente salían corriendo al enfrentarse con los guerreros humanos (este es quizá el primer caso de extras virtuales que consideran que un trabajo es demasiado peligroso).

Por supuesto, una gran cantidad de animaciones siguen siendo creadas hoy día por animadores humanos. Sin embargo, en lugar de dibujar fotogramas bidimensionales a mano como en la década de 1920, estos animadores utilizan software para manipular objetos virtuales tridimensionales dentro de un grafo de escena, de una manera que recuerda la forma de controlar las marionetas como hemos explicado anteriormente al hablar de la Cinemática. De esta forma, un animador es capaz de crear una serie de escenas virtuales que se "fotografían" para generar la animación. En algunos casos, esta técnica se emplea para generar tan solo las escenas correspondientes a los fotogramas clave y luego se emplea software para generar las escenas intermedias, generando automáticamente la escena mientras el software aplica la Dinámica y la Cinemática para desplazar los objetos del grafo de la escena entre la posición correspondiente a un fotograma clave y la siguiente.

A medida que progresan las investigaciones en el campo de los gráficos por computadora y a medida que la tecnología continúa mejorando, el proceso de animación podrá irse automatizando cada vez más. Está por ver si el papel de los animadores humanos, de los actores humanos y de los estudios físicos llegará a quedar obsoleto, pero hay mucha gente que piensa que ese día no está tan lejos. De hecho, los gráficos 3D pueden llegar a afectar a la industria del cine mucho más de lo que esta se vió afectada durante la transición del cine mudo al cine sonoro.

Cuestiones y ejercicios

1. Las imágenes que vemos los seres humanos tienden a permanecer en nuestros sistemas perceptivos durante aproximadamente 200 milise-

gundos. Basándose en esta aproximación, ¿cuántas imágenes por segundo habrá que presentar a una persona para crear una animación? Compare esta aproximación con el número de fotogramas por segundo utilizado en la industria de cine.

2. ¿Qué es un guión (***storyboard***)?
3. ¿Qué es la interpolación entre fotogramas?
4. Defina los términos *Cinemática* y *Dinámica*.

Problemas de repaso

1. ¿Cuáles de las siguientes serían aplicaciones de los gráficos 2D y cuáles de los gráficos 3D ?
   a. Diseño de la maqueta de las páginas de una revista.
   b. Dibujo de una imagen utilizando Microsoft Paint
   c. Generación de imágenes a partir de un mundo virtual para un videojuego.
2. En el contexto de los gráficos 3D, ¿qué se correspondería con cada uno de los siguientes elementos de la fotografía tradicional? Explique sus respuestas.
   a. Película.
   b. Rectángulo del visor de una cámara.
   c. La escena que se está fotografiando.
3. Al usar una proyección de perspectiva, ¿en qué condiciones una esfera de la escena no generará un círculo en el plano de proyección?
4. Al usar una proyección de perspectiva, ¿puede una imagen de un segmento de línea recta transformarse alguna vez en un segmento de línea curva en el plano de proyección? Justifique su respuesta.
5. Suponga que un extremo de una estaca recta de ocho metros de altura está situado a cuatro metros del centro de proyección. Suponga además que una línea recta trazada desde el centro de proyección a uno de los extremos de la estaca intersecta con el plano de proyección en un punto que se encuentra a un metro del centro de proyección. Si la estaca es paralela al plano de proyección, ¿qué longitud tendrá la imagen de la estaca en el plano de proyección?
6. Explique la diferencia entre una proyección paralela y una proyección en perspectiva.
7. Explique la relación entre la ventana de imagen y la memoria de fotograma.
8. Indique una diferencia importante entre la aplicación de gráficos 3D para producir una película y para generar la animación para un juego interactivo. Explique su respuesta.
9. Enumere algunas propiedades de un objeto que puedan incorporarse a un modelo de ese objeto para su uso en una escena de gráficos 3D. Identifique algunas propiedades que probablemente no se representarían en el modelo. Explique su respuesta.
10. Identifique algunas propiedades físicas de un objeto que no queden capturadas por un modelo que solo contenga una malla poligonal. (Es decir, una malla poligonal no constituye por sí sola un modelo del objeto.) Explique cómo podría añadirse una de esas propiedades al modelo del objeto.
11. ¿Podrían cuatro puntos cualesquiera en el espacio tridimensional ser los vértices de un parche que forme una malla poligonal? Explique su respuesta.
12. Cada uno de los siguientes conjuntos representa los vértices (expresados en el sistema de coordenadas rectangulares tradicional) de un parche de una malla poligonal. Describa la forma de la malla.

```
Parche 1: (0, 0, 0) (0, 2, 0)
          (2, 2, 0) (2, 0, 0)
Parche 2: (0, 0, 0) (1, 1, 1)
          (2, 0, 0)
Parche 3: (2, 0, 0) (1, 1, 1)
          (2, 2, 0)
Parche 4: (2, 2, 0) (1, 1, 1)
          (0, 2, 0)
Parche 5: (0, 2, 0) (1, 1, 1)
          (0, 0, 0)
```

13. Cada uno de los siguientes conjuntos representa los vértices (expresados en el sistema de coordenadas rectangulares tradicional) de un parche de una malla poligonal. Describa la forma de la malla.

```
Parche 1: (0, 0, 0) (0, 4, 0)
          (2, 4, 0) (2, 0, 0)
Parche 2: (0, 0, 0) (0, 4, 0)
          (1, 4, 1) (1, 0, 1)
Parche 3: (2, 0, 0) (1, 0, 1)
          (1, 4, 1) (2, 4, 0)
Parche 4: (0, 0, 0) (1, 0, 1)
          (2, 0, 0)
Parche 5: (2, 4, 0) (1, 4, 1)
          (0, 4, 0)
```

14. Diseñe una malla poligonal que represente un sólido rectangular. Utilice el sistema de coordenadas rectangulares tradicional para codificar los vértices y dibuje un boceto que represente la solución.

15. Utilizando no más de ocho parches triangulares, diseñe una malla poligonal para aproximar la forma de una esfera de radio igual a la unidad. (Con solo ocho parches, la malla será una aproximación muy burda de la esfera, pero el objetivo es ver si ha comprendido el concepto de malla poligonal, más que generar una representación precisa de una esfera.) Represente los vértices de los parches utilizando el sistema de coordenadas rectangulares tradicional y dibuje un boceto de la malla.

16. ¿Por qué los siguientes cuatro puntos no pueden ser los vértices de un parche plano?

(0, 0, 0) (1, 0, 0)
(0, 1, 0) (0, 0, 1)

17. Suponga que los puntos (1, 0, 0), (1, 1, 1) y (1, 0, 2) son los vértices de un parche plano. ¿Cuál o cuáles de los siguientes segmentos de línea son normales a la superficie del parche?

a. El segmento de línea que va de (1, 0, 0) a (1, 1, 0)
b. El segmento de línea que va de (1, 1, 1) a (2, 1, 1)
c. El segmento de línea que va de (1, 0, 2) a (0, 0, 2)
d. El segmento de línea que va de (1, 0, 0) a (1, 1, 1)

18. Identifique dos "tipos" de modelos procedimentales.

19. Compare los dos métodos más importantes de eliminación de superficies ocultas, el buffer z y el algoritmo del pintor, en lo respecta al procedimiento y a las ventajas y desventajas de cada uno de ellos.

20. ¿Cuál de las siguientes técnicas se emplea para cortar una parte de una imagen? Explique su respuesta.

a. Anti-aliasing.
b. Recorte.
c. Conversión de barrido.
d. Trazado de rayos.

21. ¿Qué es un supermuestreo? ¿Para qué se utiliza?

22. ¿Qué complicaciones introduce el hecho de que la cámara de un grafo de escena pueda cambiar su posición y su orientación?

23. Suponga que la superficie del parche plano con vértices (0, 0, 0), (0, 2, 0), (2, 2, 0) y (2, 0, 0) es suave y brillante. Si un rayo luminoso tiene su origen en el punto (0, 0, 1) e incide en la superficie en el punto (1, 1, 0), ¿a través de cuáles de los siguientes puntos pasará el rayo reflejado?

a. (0, 0, 1) b. (1, 1, 1)
c. (2, 2, 1) d. (3, 3, 1)

24. Suponga que una boya sirve de soporte a una fuente luminosa situada diez metros por encima de la superficie del agua en calma. ¿En qué punto de la superficie del agua verá un observador el reflejo de la luz,

si ese observador está a quince metros de la boya y a cinco metros por encima de la superficie del agua?

25. Si un pez está nadando por debajo de la superficie del agua, que está en calma, y un observador está contemplando al pez desde fuera del agua, ¿dónde parecerá estar el pez desde la perspectiva del observador?
a. Por encima y hacia atrás con respecto a su posición real.
b. En su posición real.
c. Por debajo y hacia delante de su posición real.

26. Suponga que los puntos (1, 0, 0), (1, 1, 1) y (1, 0, 2) son los vértices de un parche plano y que esos vértices están enumerados en sentido contrario a las agujas del reloj, vistos desde fuera del objeto. En cada uno de los siguientes casos, indique si un rayo luminoso que tenga su origen en el punto indicado incidiría sobre la superficie del parche desde el exterior o desde el interior del objeto.
a. (0, 0, 0) b. (2, 0, 0)
c. (2, 1, 1) d. (3, 2, 1)

27. Proporcione un ejemplo en el que un objeto situado fuera del volumen de visualización pudiera continuar apareciendo en la imagen final. Explique su respuesta.

28. Describa el contenido y el propósito de un buffer z.

29. ¿Cómo se generan los siguientes tipos de luz?
a. Luz ambiente.
b. Luz especular.
e. Luz difusa.

Proporcione una explicación breve.

30. Suponga que la superficie de un objeto está cubierta por franjas verticales alternativas de color naranja y azul, cada una de las cuales tiene un centímetro de anchura. Si el objeto se sitúa en una escena de modo que las posiciones de los píxeles queden asociadas con puntos del objeto a intervalos de dos centímetros, ¿cuáles serían las posibles apariencias del objeto en la imagen final? Explique su respuesta.

31. Aunque la aplicación de texturas y la aplicación de protuberancias son formas de asociar una "textura" con una superficie, son técnicas considerablemente distintas. Escriba un breve párrafo comparando las dos.

32. ¿Cómo ayuda un fotograma clave a construir un guión?

33. ¿Qué es el sangrado de color?

34. Indique en qué se diferencia el hardware de una computadora diseñada para videojuegos interactivos de un PC de propósito general?

35. ¿Qué es la captura de movimiento? ¿Dónde se utiliza?

36. ¿Es el procesamiento de imágenes un campo de los gráficos? Explique su respuesta.

37. ¿Qué es un modelo procedimental? ¿Por qué se utiliza?

38. ¿Cuáles son las ventajas de OpenGL? ¿Qué desventaja tiene?

39. ¿Qué es la eliminación de caras traseras? ¿Cuáles son las ventajas de este método?

40. Si comparamos una imagen de una escena generada mediante el trazado de rayos tradicional con una imagen similar de la misma escena producida mediante la técnica de radiosidad, ¿qué diferencias presentarán ambas imágenes?

41. ¿Cuántos fotogramas harían falta para generar una producción animada de 90 minutos para el cine?

42. Proporcione ejemplos de elementos de hardware que reduzcan la complejidad del software de aplicacion de gráficos?

43. Explique cómo ayudaría el uso de un buffer z a la hora de crear una secuencia de animación que muestre un único objeto moviéndose en una escena.

44. ¿Para qué se utilizan las curvas de Bezier?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Suponga que la animación por computadora alcanza el punto en el que ya no se necesitan actores reales en el cine ni en la televisión. ¿Cuáles serían las consecuencias? ¿Qué efectos en cascada tendría el que ya no hubiera "estrellas de cine"?
2. Con el desarrollo de las cámaras digitales y del software relacionado, la capacidad de modificar o fabricar fotografías está al alcance del público en general. ¿Qué cambios traerá esto a la sociedad? ¿Qué problemas éticos y legales podrían surgir?
3. ¿Hasta qué punto las fotografías pueden ser propiedad de alguien? Suponga que una persona carga su fotografía en un sitio web y alguna otra persona descarga esa fotografía, la modifica para que el sujeto aparezca en una situación comprometida y distribuye la versión modificada. ¿Qué podría hacer el sujeto de esa fotografía?
4. ¿Hasta qué grado es responsable el programador que ha ayudado a desarrollar un videojuego violento de las consecuencias que este pueda tener? ¿Debería restringirse el acceso de los niños a los videojuegos? En caso afirmativo, ¿cómo podría restringirse el acceso y quién debería restringirlo? ¿Y qué sucede con otros grupos sociales, como los criminales convictos?

Lecturas adicionales

Angel, E. ***Interactive Computer Graphics, A Top-Down Approach Using OpenGL***, 5ª ed. Boston, MA: Addison-Wesley, 2009.

Bowman, D. A., E. Kruijff, J. J. LaViola, Jr. y I. Poupyrev. ***3D User Interfaces Theory and Practice.*** Boston, MA: Addison-Wesley, 2005.

Hill, Jr., F. L. y S. Kelley. ***Computer Graphics Using OpenGL***. 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2007.

McConnell, J. J. ***Computer Graphics, Theory into Practice***. Sudbury, MA: Jones and Bartlett, 2006.

Parent, R. ***Computer Animation, Algorithms and Techniques***, 2ª ed. San Francisco, CA: Morgan Kaufmann, 2008.

CAPÍTULO 11

Inteligencia artificial

En este capítulo vamos a explorar la rama de las Ciencias de la computación conocida como inteligencia artificial. Aunque este campo es relativamente joven ha producido algunos resultados sorprendentes, como por ejemplo jugadores de ajedrez expertos, computadoras que parecen aprender y razonar, y máquinas que coordinan sus actividades de cara a conseguir un objetivo común, como por ejemplo ganar un partido de fútbol. En el campo de la inteligencia artificial, la ciencia ficción de hoy día podría muy bien llegar a ser la realidad de mañana.

La inteligencia artificial es el campo de las Ciencias de la computación que trata de construir máquinas autónomas, máquinas que sean capaces de llevar a cabo tareas complejas sin intervención humana. Este objetivo requiere que las máquinas sean capaces de percibir y razonar. Dichas capacidades caen dentro de la categoría de actividades de sentido común que, aunque son naturales para la mente humana, están resultando difíciles para la máquinas. El resultado es que el trabajo dentro de este campo continúa planteando grandes desafíos. En este capítulo vamos a abordar algunos de los temas de esta extensa área de investigación.

11.1 Inteligencia y máquinas

El campo de la inteligencia artificial es muy amplio y se mezcla con otros campos como la psicología, la neurología, las matemáticas, la lingüística y la ingeniería eléctrica y mecánica. Por tanto, para centrar nuestro análisis, comenzaremos considerando el concepto de agente y los tipos de comportamiento inteligente que un agente puede exhibir. De hecho, buena parte de la investigación en el campo de la inteligencia artificial puede clasificarse en función del comportamiento de un agente.

Agentes inteligentes

Un **agente** es un "dispositivo" que responde a estímulos procedentes de su entorno. Es natural pensar en un agente como si fuera una máquina individual, como por ejemplo un robot, aunque un agente puede adoptar otras formas; por ejemplo, la de un aeroplano autónomo, la de un personaje en un videojuego interactivo o la de un proceso que se comunica con otros procesos a través de Internet (quizá como cliente, como servidor o como igual, ***peer***). La mayor parte de los agentes disponen de sensores que les permiten recibir datos de su entorno; además, tienen actuadores con los que pueden ejercer una influencia sobre ese entorno. Como ejemplos de sensores podríamos citar los micrófonos, las cámaras, los sensores de proximidad y los dispositivos de muestreo del aire o del suelo. Como ejemplos de actuadores tendríamos las piernas, las alas, las pinzas y los sintetizadores de voz.

Gran parte de la investigación en inteligencia artificial puede caracterizarse en el contexto de la construcción de agentes que se comportan de manera inteligente, lo que quiere decir que las acciones de los actuadores del agente deben constituir respuestas racionales a los datos recibidos a través de sus sensores. A su vez, podemos clasificar todas esas investigaciones considerando los diferentes niveles de esas respuestas.

La respuesta más simple es una acción refleja, que es sencillamente una respuesta predeterminada a los datos de entrada. Para obtener un comportamiento más "inteligente" hacen falta niveles superiores de respuesta. Por ejemplo, podemos dotar a un agente con un conocimiento de su entorno y exigirle que ajuste sus acciones correspondientemente. El proceso de lanzar una pelota de béisbol es, en buena medida, una acción refleja pero el determinar cómo y hacia dónde lanzarla requiere un conocimiento del entorno actual (por ejemplo, saber qué jugadores están en las distintas bases del campo). En el campo de la inteligencia artificial, el problema de determinar cómo almacenar, actuali-

zar, extraer y aplicar al proceso de toma de decisiones esos conocimientos acerca del mundo real sigue planteando grandes desafíos.

Si queremos que el agente persiga un objetivo, como por ejemplo ganar una partida de ajedrez o maniobrar a través de un pasillo atestado de personas, hace falta otro nivel de respuesta aún más sofisticado. Ese comportamiento dirigido por objetivos requiere que la respuesta del agente (o su secuencia de respuestas) sea el resultado del proceso de formar deliberadamente un plan de acción o de seleccionar la mejor acción posible de entre todas las opciones existentes.

En algunos casos, las respuestas de un agente mejoran a lo largo del tiempo, a medida que el agente aprende. Esto podría adoptar la forma de un proceso de desarrollo de un **conocimiento procedimental** (aprender "cómo") o de un proceso de almacenamiento de **conocimiento declarativo** (aprender "qué"). El aprendizaje de conocimiento procedimental suele implicar un proceso de prueba y error, mediante el que el agente aprende las acciones apropiadas a base de ser castigado por las acciones incorrectas y premiado por las que son acertadas. De acuerdo con este enfoque, se han desarrollado agentes que mejoran a lo largo del tiempo sus habilidades en juegos de competición, como las damas y el ajedrez. El aprendizaje de conocimiento declarativo suele consistir en ampliar o modificar los "hechos" presentes en el almacén de conocimiento de un agente. Por ejemplo, un jugador de béisbol debe ajustar repetidamente su base de datos de conocimiento (por ejemplo, para reflejar la posición cambiante de los jugadores en las distintas bases del campo) a partir de la cual se determinan las respuestas racionales a los sucesos futuros.

Para producir respuestas racionales a los estímulos, un agente debe "comprender" los estímulos recibidos a través de sus sensores. Es decir, el agente tiene que ser capaz de extraer información de los datos que sus sensores generan o, en otras palabras, el agente debe ser capaz de percibir. En algunos casos, se trata de un proceso sencillo. Las señales obtenidas de un giróscopo deben codificarse en formas que sean compatibles con los cálculos necesarios para determinar las respuestas. Pero en otros casos, la extracción de información de los datos de entrada resulta difícil. Como ejemplos podemos citar los intentos de comprender el lenguaje y las imágenes. Del mismo modo, los agentes tienen que ser capaces de formular sus respuestas en términos compatibles con los actuadores de los que disponen. Puede que esto sea un proceso sencillo, o ese proceso puede exigir al agente que formule las respuestas como frases completas habladas, lo que significa que el agente debe ser capaz de expresarse de forma oral. Por ello, temas tales como el procesamiento y análisis de imágenes, la comprensión del lenguaje natural y la generación del habla son áreas de investigación de gran importancia.

Los atributos de los agentes que hemos identificado representan áreas de investigación tanto actuales como pasadas. Por supuesto, no son completamente independientes entre sí. Nos gustaría poder desarrollar agentes que poseyeran todos esos atributos, obteniendo así agentes que comprendieran los datos recibidos de su entorno y desarrollaran nuevos patrones de respuesta a través de un proceso de aprendizaje, cuyo objetivo sería maximizar las habilidades del agente. Sin embargo, aislando los diversos tipos de comportamiento racional y tratando de obtenerlos de manera independiente, los investigadores obtienen avances que pueden combinarse posteriormente con los progresos realizados en otras áreas, con el fin de producir agentes más inteligentes.

Terminamos esta subsección presentando un agente que proporcionará el contexto para las explicaciones contenidas en las Secciones 11.2 y 11.3. El agente está diseñado para resolver el puzzle de los ocho cuadrados, el cual está compuesto por ocho piezas cuadradas etiquetadas del 1 al 8 y montadas en un marco capaz de albergar un número total de nueve de esas piezas, dispuestas en tres filas y tres columnas (Figura 11.1). Entre las piezas situadas en el marco hay un hueco al que puede moverse cualquiera de las piezas adyacentes, lo que permite ir mezclando las piezas del marco. El problema consiste en mover las piezas otra vez a sus posiciones iniciales (Figura 11.1) partiendo de una disposición desordenada de las mismas.

Nuestro agente adoptará la forma de una caja equipada con una pinza, una cámara de vídeo y un dedo con un extremo recubierto de goma, para que no se deslice al empujar algún objeto (Figura 11.2). Cuando se enciende el agente por primera vez, su pinza se abre y se cierra, como si estuviera pidiendo el puzzle. Cuando colocamos un puzzle con las ocho piezas desordenadas en la pinza, esta se cierra para sujetarlo. Después de un breve periodo de tiempo, el dedo de la máquina desciende y comienza a empujar las piezas de un lado a otro dentro del marco, hasta colocarlas en su posición original. Llegados a este punto, la máquina suelta el puzzle y se apaga automáticamente.

Esta máquina solucionadora de puzzles exhibe dos de los atributos de los agentes que hemos identificado anteriormente. En primer lugar, debe ser capaz de percibir, en el sentido de que debe extraer el estado actual del puzzle de la imagen recibida a través de la cámara. Consideraremos la cuestión de la compresión de imágenes en la Sección 11.2. En segundo lugar, la máquina debe desarrollar e implementar un plan para conseguir el objetivo. Consideraremos estas cuestiones en la Sección 11.3.

Metodologías de investigación

Para orientarse en el campo de la inteligencia artificial, es útil entender que este campo está avanzando en dos direcciones distintas. Una de las líneas de avance es la ingenieril, en la que los investigadores están tratando de desarrollar sistemas que exhiban un comportamiento inteligente. La otra línea de avance es la teórica, en la que los investigadores tratan de desarrollar una comprensión computacional de la inteligencia animal, y especialmente de la inteligencia humana. Podemos clarificar esta dicotomía considerando la manera en la que se persiguen esos dos objetivos. La técnica ingenieril conduce a una metodología orientada al rendimiento, porque el objetivo último es construir un producto que satisfaga ciertos objetivos de rendimiento. El enfoque teórico

Figura 11.1 El puzzle de ocho piezas una vez resuelto.

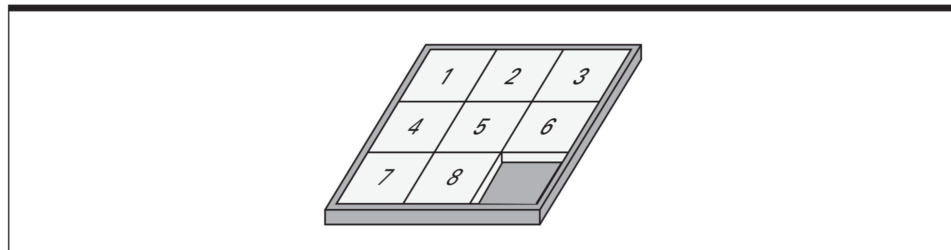
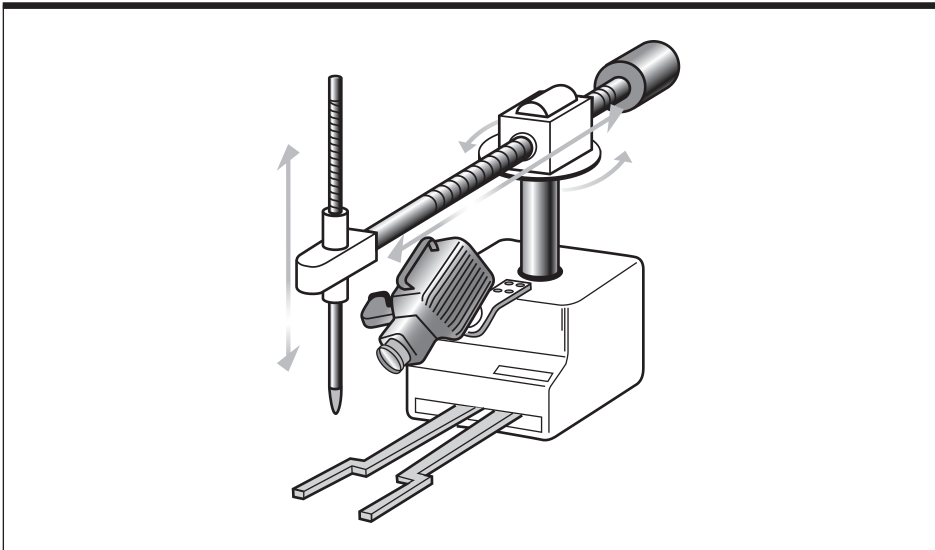


Figura 11.2 Nuestra máquina solucionadora de puzzles.



conduce a una metodología orientada a la simulación, porque el objetivo último es ampliar nuestra comprensión de la inteligencia, por lo que el énfasis se pone en el proceso subyacente, más que en el comportamiento exterior.

Por ejemplo, considere los campos del procesamiento del lenguaje natural y de la lingüística. Estos dos campos están íntimamente relacionados y ambos se benefician de las investigaciones realizadas en el otro, a pesar de lo cual los objetivos últimos son diferentes. Los lingüistas están interesados en aprender cómo procesan el lenguaje los seres humanos, por lo que tratan de alcanzar objetivos más teóricos. Los investigadores en el campo del procesamiento del lenguaje natural están interesados en desarrollar máquinas que puedan manipular el lenguaje natural, por lo que sus objetivos tienden a adoptar una orientación ingenieril. Por ello, los lingüistas operan en un modo orientado a la simulación, construyendo sistemas cuyo objetivo es el de probar teorías. Por el contrario, los investigadores en el campo del procesamiento del lenguaje natural operan en un modo orientado al rendimiento, construyendo sistemas con el fin de realizar tareas. Los sistemas construidos de este último modo (como por ejemplo los traductores de documentos y los sistemas que permiten a las máquinas responder a las órdenes verbales) dependen en gran medida del conocimiento obtenido por los lingüistas, pero a menudo aplican "atajos" que resulta que funcionan adecuadamente en el entorno restringido en el que opera cada sistema concreto.

Como ejemplo elemental, considere la tarea de desarrollar un intérprete de órdenes (*shell*) para un sistema operativo que reciba instrucciones del mundo exterior a través de comandos verbales en español. En este caso, la shell (un agente) no necesita preocuparse de todo el idioma español. Para ser más precisos, la shell no necesita distinguir entre los distintos significados de la palabra ***copiar*** (¿Es un nombre o un verbo? ¿Debe incluir la connotación de plagio?). En lugar de ello, la shell necesita simplemente distinguir la palabra ***copiar*** de otros comandos como ***renombrar*** y ***borrar***. Por tanto, esa shell podría realizar su

tarea simplemente viendo si sus entradas se corresponden con patrones de audio predeterminados. El rendimiento de un sistema de este tipo podría ser satisfactorio para un ingeniero, pero la forma en que se obtiene ese comportamiento no resultaría estéticamente placentera para un teórico.

El test de Turing

En el pasado, el **test de Turing** (propuesto por Alan Turing en 1950) ha servido como manera de medir el progreso en el campo de la inteligencia artificial. Hoy día, la importancia que se concedía al test de Turing se ha desvanecido, aunque continúa formando parte del folklore propio del campo de la inteligencia artificial. La propuesta de Turing consistía en permitir que una persona, a la que llamaremos interrogador, se comunicara con un sujeto de prueba por medio de un teclado, sin que el interrogador supiera si el sujeto de prueba era un ser humano o una máquina. En este entorno, diríamos que una máquina se comporta de manera inteligente si el interrogador no es capaz de distinguirla de un ser humano. Turing predijo que para el año 2000 las máquinas tendrían un 30 por ciento de posibilidades de pasar un test de Turing de cinco minutos de duración, una conjetura que ha resultado ser sorprendentemente precisa.

Una de las razones por la que ya no se considera el test de Turing como una medida significativa de la inteligencia es que se puede **producir** con facilidad una inquietante apariencia de inteligencia. Un ejemplo muy conocido surgió como resultado del programa DOCTOR (una versión del sistema más general denominado ELIZA) desarrollado por Joseph Weizenbaum a mediados de la década de 1960. Este programa interactivo estaba diseñado para proyectar la imagen de un analista rogeriano realizando una sesión de psicoterapia. La computadora desempeñaba el papel del analista, mientras que el usuario hacía de paciente. Internamente, lo único que DOCTOR hacía era reestructurar las frases pronunciadas por el paciente de acuerdo con algunas reglas bien definidas y volver a plantearlas al paciente. Por ejemplo, en respuesta a la frase "Hoy estoy cansado", DOCTOR podría haber contestado "¿Por qué cree que está cansado hoy?". Si DOCTOR era incapaz de reconocer la estructura de la frase, simplemente respondía con algo como "Continúe" o "Eso es muy interesante".

Los orígenes de la inteligencia artificial

Los intentos de construir máquinas que imiten el comportamiento humano son muy antiguos, pero muchos expertos coincidirán en que el campo moderno de la inteligencia artificial tiene su origen en 1950. Ese es el año en el que Alan Turing publicó el artículo "*Computing Machinery and Intelligence*" (máquinas de computación e inteligencia) en el que proponía que podrían programarse máquinas que exhibieran comportamiento inteligente. El nombre del campo, *inteligencia artificial,* fue acuñado unos cuantos años después en la ahora legendaria propuesta escrita por John McCarthy, en la que sugería que "se llevara a cabo un estudio sobre inteligencia artificial durante el verano de 1956 en el Dartmouth College" para explorar "la conjetura de que cualquier aspecto del aprendizaje o cualquier otra característica de la inteligencia puede ser, en principio, descrita de forma tan precisa que es posible construir una máquina para simularla."

El propósito de Weizenbaum al desarrollar DOCTOR era el estudio de la comunicación en lenguaje natural. El tema de la psicoterapia simplemente proporcionaba un entorno en el que el programa podía "comunicarse". Sin embargo, para consternación de Weizenbaum, varios psicólogos propusieron utilizar su programa en psicoterapias reales. (La tesis rogeriana es que es el paciente, y no el psicoanalista, el que debe conducir la conversación durante la terapia, así que esos psicólogos argumentaban que una computadora podría posiblemente participar en la conversación tan bien como podría hacerlo un terapeuta.) Además, DOCTOR proyectaba una imagen de comprensión tan fuerte que muchos de los que se "comunicaban" con el programa llegaban a aficionarse a esos diálogos de preguntas y respuestas que mantenían con la máquina. En cierto sentido, DOCTOR pasó el test de Turing. El resultado fue que se plantearon cuestiones tanto éticas como técnicas y Weizenbaum terminó por convertirse en un activo defensor de la tesis de que hay que mantener la dignidad humana en este mundo de avances tecnológicos.

Otros ejemplos más recientes de "éxitos" a la hora de pasar el test de Turing serían los virus de Internet que llevan a cabo diálogos "inteligentes" con una víctima humana, para convencerla de que desactive sus protecciones contra el *malware*. Además, fenómenos similares a los test de Turing son los que se producen en el contexto de determinados juegos de computadora, como por ejemplo los programas para jugar al ajedrez. Aunque estos programas seleccionan sus movimientos aplicando simplemente técnicas de fuerza bruta (similares a las que explicaremos en la Sección 11.3), las personas que compiten contra la computadora experimentan a menudo la sensación de que las máquinas poseen creatividad e incluso una personalidad. Sensaciones similares se experimentan en el campo de la robótica, en el que se han construido máquinas con atributos físicos que permiten proyectar características inteligentes. Como ejemplos podemos citar los perros robot de juguete, que proyectan personalidades adorables simplemente moviendo la cabeza o levantando las orejas en respuesta a un sonido.

Cuestiones y ejercicios

1. Identifique diversos tipos de acciones "inteligentes" que un agente podría llevar a cabo.
2. Una planta colocada en una habitación oscura en la que solo exista una fuente luminosa tenderá a crecer hacia la luz. ¿Se trata de una respuesta inteligente? ¿Posee inteligencia la planta? ¿Cómo definiría entonces la inteligencia?
3. Suponga que diseñamos una máquina expendedora para suministrar varios productos dependiendo del botón que se pulse. ¿Diría que esa máquina es "consciente" de qué botón se ha pulsado? ¿Cómo definiría entonces el hecho de "ser consciente"?
4. Si una máquina pasa el test de Turing, ¿estaría de acuerdo en afirmar que es inteligente? En caso negativo, ¿estaría de acuerdo en afirmar que parece inteligente?

5. Suponga que utiliza un programa de mensajería para hablar con alguien a través de Internet y que mantiene con ese interlocutor una conversación coherente y con sentido durante diez minutos. Si posteriormente averiguara que ha estado conversando con una máquina, ¿concluiría que esa máquina es inteligente? Explique su respuesta.

11.2 Percepción

Para responder de manera inteligente a las entradas procedentes de sus sensores, un agente debe ser capaz de comprender esas entradas. Es decir, el agente tiene que ser capaz de percibir. En esta sección vamos a explorar dos áreas de investigación en el campo de la percepción que han demostrado ser especialmente complicadas: la comprensión de las imágenes y del lenguaje.

Comprensión de las imágenes

Consideremos los problemas planteados por la máquina solucionadora de puzzles que hemos presentado en la sección anterior. La apertura y cierre de la pinza de la máquina no supone obstáculo de importancia y la capacidad de detectar la presencia del puzzle en la pinza durante este proceso es bastante sencilla de implementar, porque nuestra pinza requiere muy poca precisión. Incluso el problema de centrar la cámara sobre el puzzle puede solventarse diseñando simplemente la pinza de modo que coloque el puzzle en una posición predeterminada que resulte adecuada para su visualización. En consecuencia, el primer comportamiento inteligente que la máquina requiere es la extracción de información a través de un medio visual.

Es importante comprender que el problema al que se enfrenta nuestra máquina a la hora de examinar el puzzle no es el de simplemente producir y almacenar una imagen. La tecnología es capaz de hacer esto desde hace muchos años, como por ejemplo en los sistemas tradicionales de fotografía y televisión. Más bien, el problema consiste en comprender la imagen con el fin de extraer de ella el estado actual del puzzle (y quizá, posteriormente, monitorizar el movimiento de las piezas).

En el caso de nuestra máquina solucionadora de puzzles, las posibles interpretaciones de la imagen de un puzzle son relativamente limitadas. Podemos suponer que lo que aparece es siempre una imagen que contiene los dígitos 1 a 8 en un patrón bien organizado. El problema consiste simplemente en extraer la disposición de estos dígitos. Para ello, vamos a imaginar que la imagen del puzzle se ha codificado en forma de bits en la memoria de la computadora, representando cada bit el nivel de brillo de un píxel concreto. Asumiendo que el tamaño de la imagen es uniforme (la máquina sostiene el puzzle en una posición predeterminada delante de la cámara), nuestra máquina puede detectar qué pieza se encuentra en cada posición por el procedimiento de comparar las diferentes secciones de la imagen con una serie de plantillas pregrabadas, que estarán compuestas por los patrones de bits generados por los dígitos individuales que se usan en el puzzle. A medida que vayan detectándose correspondencias, se irá revelando el estado actual del puzzle.

Esta técnica de reconocimiento de imágenes es un método utilizado en los lectores ópticos de caracteres. Sin embargo, tiene la desventaja de exigir un cierto grado de uniformidad en cuanto al estilo, el tamaño y la orientación de los símbolos que se están leyendo. En particular, el patrón de bits generado por un carácter físicamente grande no se ajusta a la plantilla correspondiente a una versión más pequeña del mismo símbolo, incluso aunque las formas sean iguales. Además, es fácil imaginar hasta qué punto se incrementa la dificultad del problema a la hora de intentar procesar material manuscrito.

Otra solución al problema del reconocimiento de caracteres se basa en intentar encontrar correspondencias entre las características geométricas en lugar de buscar correspondencias relativas a la apariencia exacta de los símbolos. En este caso, el dígito 1 podría caracterizarse como una única línea vertical, el 2 podría considerarse una línea curva abierta unida a una línea recta horizontal por su parte inferior, y así sucesivamente. Este método de reconocimiento de símbolos consta de dos etapas: la primera consiste en extraer las características de la imagen que se está procesando y la segunda en comparar esas características con la de los símbolos conocidos. Como sucede con la técnica basada en la detección de correspondencias con plantillas pregrabadas, esta técnica de reconocimiento de caracteres tampoco es perfecta. Por ejemplo, una serie de errores poco importantes en la imagen pueden generar un conjunto de características geométricas completamente distintas, como sucede al tratar de distinguir entre una O y una C o, si hablamos del puzzle de ocho piezas, al tratar de distinguir el 3 del 8.

Nosotros tenemos suerte en nuestra aplicación del puzzle porque no necesitamos comprender imágenes de escenas tridimensionales genéricas. Considere, por ejemplo, la ventaja que tenemos al poder estar seguros de que las formas que hay que reconocer (los dígitos de 1 a 8) están aisladas en diferentes partes de la imagen, en lugar de aparecer como imágenes solapadas, que es lo que sucede comúnmente en la mayoría de los escenarios de carácter general. Por ejemplo, en una fotografía genérica, no solo tenemos el problema de reconocer un objeto desde diferentes ángulos, sino que también tenemos que enfrentarnos con el hecho de que algunas partes del objeto pueden estar ocultas a la vista.

La tarea de comprender imágenes genéricas suele abordarse mediante un proceso en dos pasos: (1) **procesamiento de imágenes**, que hace referencia a la tarea de identificar las características de la imagen y (2) **análisis de imágenes**, que hace referencia a la tarea de comprender lo que esas características significan. Ya hemos observado esta dicotomía en el contexto de reconocimiento de símbolos mediante sus características geométricas. En esa situación, el procesamiento de imágenes está representado por el proceso de identificar las características geométricas presentes en la imagen, mientras que el análisis de imágenes está representado por el proceso de identificar el significado de esas características.

El procesamiento de imágenes abarca numerosos temas distintos. Uno de ellos es el del realce de contornos, que es el proceso de aplicar técnicas matemáticas para clarificar las fronteras entre distintas regiones de una imagen. En cierto sentido, el realce de contorno es un intento de convertir una fotografía en un dibujo compuesto por líneas. Otra de las actividades dentro del procesamiento de imágenes se conoce con el nombre de localización de regiones. Este

Inteligencia artificial fuerte e inteligencia artificial débil

La conjetura de que las máquinas pueden programarse para exhibir comportamiento inteligente se conoce como **inteligencia artificial débil** y es aceptada, en distinto grado, por un gran número de expertos en la actualidad. Sin embargo, la conjetura de que las máquinas pueden programarse para poseer inteligencia y, de hecho, consciencia, conjetura que se conoce con el nombre de **inteligencia artificial fuerte**, es objeto de grandes debates. Los oponentes de la inteligencia artificial fuerte argumentan que una máquina es inherentemente distinta de un ser humano y que por tanto jamás puede sentir amor, diferenciar el bien del mal y pensar acerca de sí misma de la misma manera que un ser humano lo hace. Sin embargo, los defensores de la inteligencia artificial fuerte argumentan que la mente humana está construida a partir de pequeños componentes que, considerados individualmente, no son humanos y tampoco conscientes, pero que al combinarse, sí lo son. ¿Por qué no podría darse el mismo fenómeno en las máquinas, argumentan?

El problema a la hora de resolver el debate de la inteligencia artificial fuerte es que atributos tales como la inteligencia y la consciencia son características internas que no pueden identificarse directamente. Como Alan Turing señaló, otorgamos inteligencia a otros seres humanos porque se comportan de manera inteligente, aún cuando no seamos capaces de observar sus estados mentales internos. ¿Estamos entonces preparados para actuar igual con las máquinas, si estas exhiben las características externas de la consciencia? ¿Por qué?

es el proceso de identificar aquellas áreas de la imagen que tienen propiedades comunes, como por ejemplo el mismo brillo, color o textura. Esas regiones probablemente representen secciones de la imagen que pertenecen a un mismo objeto. (Es la capacidad de reconocer regiones lo que permite a las computadoras añadir color a las películas antiguas en blanco y negro.) Otra actividad más dentro del ámbito del procesamiento de imágenes es el suavizado, que es el proceso de eliminar fallos en la imagen. El suavizado evita que los errores de la imagen confundan a los otros pasos del procesamiento de imágenes, aunque una cantidad excesiva de suavizado puede hacer que se pierda también información importante.

El suavizado, el realce de contornos y la localización de regiones son pasos del proceso de identificación de los distintos componentes de una imagen. El análisis de imágenes es el proceso de determinar qué es lo que esos componentes representan y, en último término, qué es lo que la imagen significa. Aquí nos encontramos con problemas tales como reconocer desde diferentes perspectivas objetos cuya visión está parcialmente obstruida. Una técnica de análisis de imágenes consiste en comenzar con una suposición acerca de lo que puede ser la imagen y luego tratar de asociar los componentes de la imagen con los objetos cuya presencia se conjetura. Esta parece ser una solución que los seres humanos aplicamos. Por ejemplo, en ocasiones nos resulta difícil reconocer un objeto inesperado en un entorno en el que nuestra visión esté desenfocada, pero en cuanto tenemos una pista acerca de lo que el objeto puede ser, podemos identificarlo fácilmente.

Los problemas asociados con el análisis de imágenes genéricas son enormes y todavía quedan por hacer muchas investigaciones en este área. De hecho, el análisis de imágenes es uno de esos campos que ilustra cómo muchas

tareas que la mente humana realiza de forma rápida y aparentemente fácil continúan desafiando las capacidades de las máquinas.

Procesamiento del lenguaje

Otro problema de percepción que ha demostrado ser difícil de resolver es el de la comprensión del lenguaje. El éxito obtenido al traducir lenguajes de programación de alto nivel a lenguaje máquina (Sección 6.4) hizo pensar a los primeros investigadores que la capacidad de programar a las computadoras para que comprendieran el lenguaje natural se podría conseguir en solo unos pocos años. De hecho, la capacidad de traducir programas proporciona la ilusión de que la máquina comprende realmente el lenguaje que se está traduciendo. (Recuerde de la Sección 6.1 la historia de Grace Hopper acerca de esos directivos que creían que estaba enseñando a las computadoras a entender el alemán.)

Lo que esos investigadores no comprendían eran las enormes diferencias entre los lenguajes de programación formales y los lenguajes naturales como el español, el inglés, el alemán o el latín. Los lenguajes de programación se construyen a partir de primitivas bien diseñadas, de modo que cada sentencia tiene una única estructura gramatical y un único significado. Por el contrario, una sentencia en un lenguaje natural puede tener múltiples significados, dependiendo del contexto o incluso de la manera en que sea comunicada. Por ello, para comprender el lenguaje natural, los humanos dependen fuertemente de la posesión de conocimientos adicionales.

Por ejemplo, las frases

Velázquez pintaba personas.

y

Juan es un burro.

tienen varios significados, que no pueden distinguirse traduciendo o analizando sintácticamente cada palabra de forma independiente. En lugar de ello, hace falta la capacidad de comprender el contexto en el que se dice la frase. En otros casos, el verdadero significado de una frase no coincide con su traducción literal. Por ejemplo,

¿Sabes qué hora es?

significa a menudo “Dime qué hora es, por favor”, o si quien la dice ha estado esperando durante un buen rato, puede querer decir “Llegas muy tarde”.

Para descubrir el significado de una frase en el lenguaje natural, se requieren por tanto varios niveles de análisis. El primero de ellos es el **análisis sintáctico**. Es en este nivel cuando se reconoce a *María* como sujeto de la frase

María le dió a Juan una tarjeta de cumpleaños.

mientras que en la frase

Juan recibió una tarjeta de cumpleaños de María.

se reconoce que el sujeto es *Juan.*

Otro nivel de análisis es el **análisis semántico**. A diferencia del proceso de análisis sintáctico, que simplemente identifica el papel gramatical de cada palabra, el análisis semántico tiene la tarea de identificar el papel semántico de

cada palabra de la frase. El análisis semántico trata de identificar cosas tales como la acción descrita, el agente de esa acción (que puede ser o no el sujeto de la frase) y el objeto de la acción. Es el análisis semántico lo que permite saber que las frases "María le dió a Juan una tarjeta de cumpleaños" y "Juan recibió una tarjeta de cumpleaños de María" dicen lo mismo.

Un tercer nivel de análisis es el **análisis contextual**. Es en este nivel cuando se incluye en el proceso de comprensión el contexto de la frase. Por ejemplo, es fácil identificar el papel gramatical de cada palabra en la frase

El actor rechazó el papel.

Podemos incluso realizar un análisis semántico identificando la acción implicada como ***rechazó,*** el agente como ***actor,*** y así sucesivamente. Pero hasta que no consideramos el contexto de la frase, no queda claro el significado de la misma. En concreto, tendrá un significado distinto si la palabra ***papel*** hace referencia a un personaje de una obra teatral o a una hoja de papel. Precisamente, es en el nivel contextual cuando podrá revelarse finalmente el verdadero significado de la pregunta "¿Sabes qué hora es?".

Hay que recalcar que los diversos niveles de análisis (sintáctico, semántico y contextual) no son necesariamente independientes entre sí. Por ejemplo, imagine que el pie de una fotografía dice

Detenidos comiendo bocadillos.

En este caso, las personas que aparecen en la fotografía pueden ser el sujeto de la frase (los ***detenidos***) si lo que la fotografía muestra es a unas personas que habían sido anteriormente detenidas y que ahora están comiéndose unos bocadillos. Sin embargo, las personas que aparecen en la fotografía serán el objeto si lo que la fotografía muestra son unas personas que están siendo detenidas mientras se comen un bocadillo. Por tanto, la frase tiene más de una estructura gramatical posible y cuál de las dos sea correcta dependerá del contexto, es decir, de lo que la fotografía muestre.

Otra área de investigación en el procesamiento del lenguaje natural es la que se refiere a documentos completos en lugar de a frases individuales. En este caso, los problemas caen dentro de dos categorías distintas: **recuperación de información** y **extracción de información**. La recuperación de información hace referencia a la tarea de identificar documentos relacionados con el tema que estemos tratando. Un ejemplo sería el problema al que se enfrentan los usuarios de la World Wide Web al tratar de buscar sitios relacionados con un tema concreto. La solución más popular consiste en buscar sitios de acuerdo con una serie de palabras clave, pero esto genera a menudo una avalancha de sitios carentes de interés, al mismo tiempo que puede no proporcionar ningún enlace que sí tiene importancia simplemente por el hecho de que en ese sitio se utiliza "automóvil" en lugar de "coche" o "vehículo". Lo que hace falta es un mecanismo de búsqueda que comprenda el contenido de los sitios que se analizan. La dificultad de obtener ese tipo de comprensión es la razón por la que muchos investigadores están adoptando técnicas como XML con el fin de obtener una Web semántica, como ya hemos indicado en la Sección 4.3.

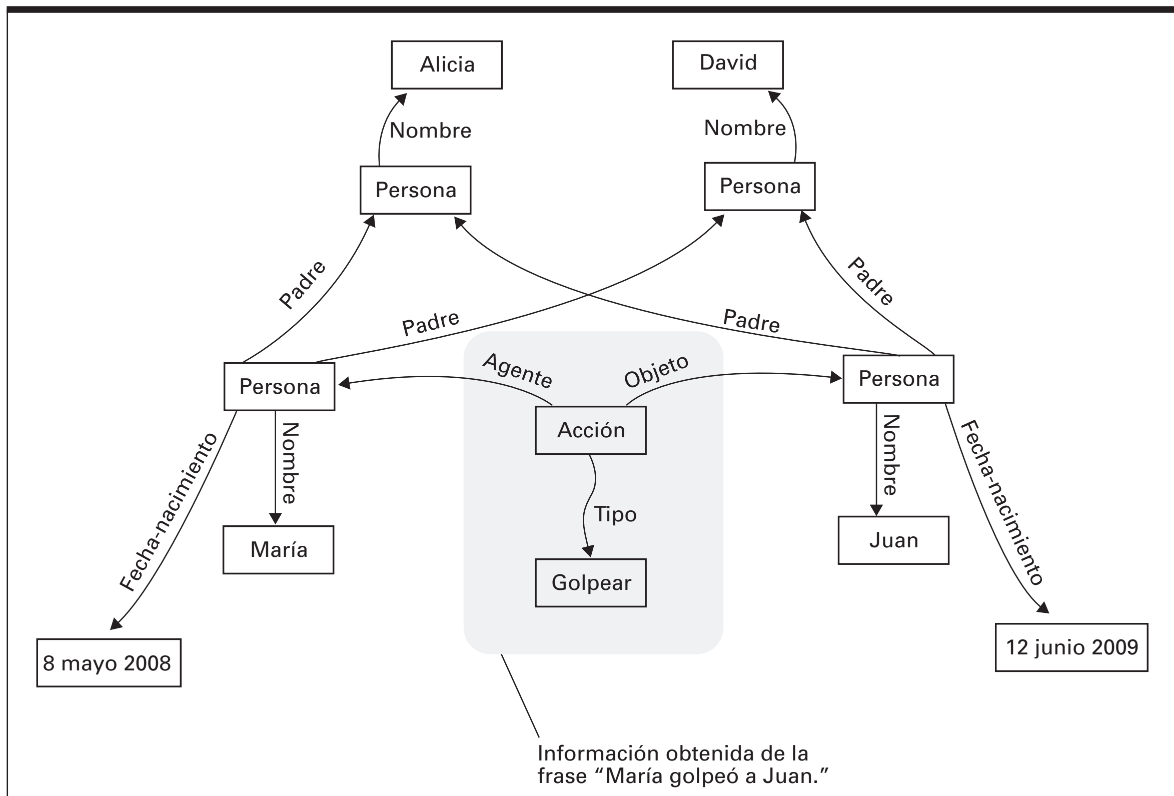
La extracción de información hace referencia a la tarea de extraer información de los documentos, con el fin de darle una forma que resulte útil para otras aplicaciones. Esto puede implicar identificar la respuesta a una cuestión

específica o guardar la información de manera tal que puedan responderse cuestiones en algún momento posterior. Una de esas maneras en las que puede almacenarse la información se conoce con el nombre de **marco**, que es básicamente una plantilla en la que se guardan detalles específicos. Por ejemplo, considere un sistema para leer un periódico. El sistema podría utilizar diversos marcos, uno por cada tipo de artículo que pueda aparecer en un periódico. Si el sistema identifica que un artículo trata sobre un robo, intentará rellenar las casillas necesarias dentro del marco correspondiente a los robos. Este marco solicitaría, probablemente, elementos de información tales como la dirección del robo, la fecha y la hora en que se produjo, los objetos que fueron sustraídos, etc. Por el contrario, si el sistema identifica que el artículo informa sobre un desastre natural, rellenará el marco correspondiente a los desastres naturales, que indicará al sistema que debe identificar el tipo de desastre, los daños producidos, etc.

Otra manera en la que los extractores de información guardan la información extraída se conoce con el nombre de **red semántica**. Se trata, esencialmente, de una gran estructura de datos enlazados, en la que se emplean punteros para indicar las asociaciones entre los distintos elementos de datos. La Figura 11.3 muestra parte de una red semántica en la que se ha resaltado la información obtenida a partir de la frase

María golpeó a Juan.

Figura 11.3 Una red semántica.

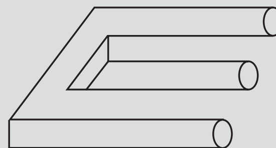


Inteligencia artificial en la palma de la mano

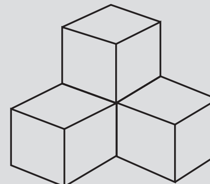
Las técnicas de inteligencia artificial están incorporándose cada vez más en las aplicaciones para teléfonos inteligentes. Por ejemplo, Google ha desarrollado Google Goggles, una aplicación para teléfono inteligente que proporciona un motor de búsqueda visual. Basta con hacer una fotografía de un libro, de un paisaje o de un anuncio utilizando la cámara del teléfono y Goggles realiza el procesamiento de imágenes, el análisis de imágenes y el reconocimiento de textos, después de lo cual inicia una búsqueda en la Web para identificar el objeto. Por ejemplo, un turista inglés que esté de visita en Francia puede hacer una fotografía de una señal de tráfico, de un menú o de cualquier otro texto y hacer que la aplicación la traduza al inglés. Además de en Goggles, Google está trabajando activamente en el campo de la traducción de idiomas voz a voz. Muy pronto seremos capaces de hablar en español a través de nuestro teléfono y hacer que esas palabras sean traducidas automáticamente al inglés, al chino o a cualquier otro idioma. Los teléfonos inteligentes irán teniendo cada vez más capacidades a medida que se vaya utlizando la tecnología de la inteligencia artificial en formas cada vez más innovadoras.

Cuestiones y ejercicios

1. ¿Cómo difieren los requisitos de un sistema de vídeo de un robot si el robot lo utiliza para controlar sus actividades, frente al caso en que esa información recopilada por el sistema de vídeo se transmita a una persona que sea quien controle remotamente el robot?
2. ¿Qué es lo que nos dice que el siguiente dibujo no tiene sentido? ¿Cómo podría programarse en una máquina ese tipo de razonamiento?



3. ¿Cuántos bloques hay en la pila de bloques que se muestra a continuación? ¿Cómo podría programarse una máquina para responder a este tipo de preguntas de forma correcta?



4. ¿Cómo sabemos que las dos frases “Nada es mejor que una felicidad completa” y “Un tazón de sopa es mejor que nada” no implican que “un tazón de sopa es mejor que una felicidad completa”? ¿Cómo podríamos transferir a una máquina nuestra capacidad de realizar esta diferenciación?

5. Identifique las ambigüedades que aparecen al traducir la frase "Han puesto un banco nuevo en la plaza".
6. Compare los resultados de analizar sintácticamente las dos frases siguientes. A continuación, explique cómo difieren estas frases semánticamente.

   El granjero construyó la cerca en el campo.

   El granjero construyó la cerca en el verano.
7. Basándose en la red semántica de la Figura 11.3, ¿cuál es la relación familiar entre María y Juan?

11.3 Razonamiento

Utilicemos ahora la máquina solucionadora de puzzles presentada en la Sección 11.1 para explorar las técnicas para el desarrollo de agentes dotados de capacidad de razonamiento elemental.

Sistemas de producción

Una vez que nuestra máquina solucionadora de puzzles ha descifrado las posiciones de las piezas a partir de la imagen visual, tiene que tratar de averiguar qué movimientos hay que hacer para resolver el puzzle. Una solución a este problema que se nos podría ocurrir es preprogramar la máquina con soluciones para todas las posibles disposiciones de las piezas. Entonces la tarea de la máquina consistiría simplemente en seleccionar y ejecutar el programa apropiado. Sin embargo, el puzzle de ocho piezas tiene más de 100.000 configuraciones posibles, por lo que la idea de proporcionar una solución explícita para cada una no es muy tentadora. Por tanto, nuestro objetivo será programar la máquina de manera que pueda construir soluciones para el puzzle por su cuenta. Es decir, la máquina debe programarse para realizar actividades de razonamiento básicas.

El desarrollo de las capacidades de razonamiento en un máquina ha sido objeto de investigación durante muchos años. Uno de los resultados de estas investigaciones es el reconocimiento de que existe una gran clase de problemas de razonamiento que presentan características comunes. Dichas características se pueden definir mediante una entidad abstracta conocida con el nombre de **sistema de producción**, que está formada por tres componentes principales:

1. *Un conjunto de estados.* Cada **estado** es una situación que puede presentarse dentro del entorno de aplicación concreto en el que nos encontremos. El estado del que se parte se denomina **estado inicial**; el estado (o estados) deseado se denomina **estado objetivo**. En nuestro caso, un estado es una de las posibles configuraciones del puzzle de ocho piezas; el estado inicial es la configuración del puzzle en el momento de entregárselo a la máquina y el estado objetivo es la configuración del puzzle resuelto, tal como se muestra en la Figura 11.1.
2. *Un conjunto de producciones (reglas o movimientos).* Una **producción** es una operación que puede realizarse en el entorno de aplicación con el

fin de pasar de un estado a otro. Cada producción puede estar asociada con una serie de precondiciones; es decir, pueden existir condiciones que deban estar presentes obligatoriamente en el entorno para que una determinada producción pueda aplicarse. (En nuestro caso, las producciones son los movimientos de las piezas. Cada movimiento de una pieza tiene la precondición de que el hueco debe estar situado al lado de la pieza en cuestión.)

3. *Un sistema de control*. El **sistema de control** está compuesto de la lógica que resuelve el problema de pasar del estado inicial al estado objetivo. En cada paso del proceso, el sistema de control debe decidir qué producción aplicar a continuación de entre todas las producciones cuyas precondiciones estén satisfechas. (Dado un estado concreto en nuestro puzzle de ocho piezas, habrá varias piezas colocadas junto al hueco y por tanto, varias producciones aplicables. El sistema de control debe decidir qué pieza mover.)

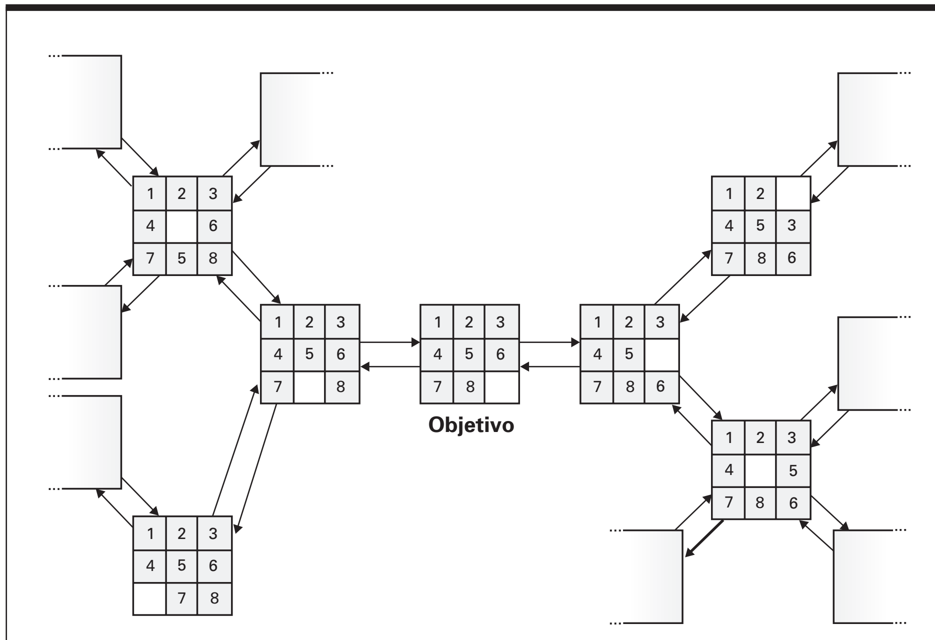
Observe que la tarea asignada a nuestra máquina solucionadora de puzzles puede formularse en el contexto de un sistema de producción. En este tipo de entorno, el sistema de control adopta la forma de un programa. Este programa inspecciona el estado actual del puzzle, identifica una secuencia de producciones que lleve hasta el estado objetivo y ejecuta esa secuencia. Es tarea nuestra, por tanto, diseñar un sistema de control para resolver el puzzle de ocho piezas.

Un concepto importante en el desarrollo de un sistema de control es el de **espacio del problema**, que es el conjunto de todos los estados, producciones y precondiciones existentes en un sistema de producción. Un espacio del problema suele conceptualizarse a menudo en la forma de un **grafo de estados**. Aquí, el término *grafo* hace referencia a una estructura que los matemáticos denominarían **grafo dirigido** y que no es otra cosa que un conjunto de ubicaciones denominadas **nodos** conectados mediante flechas. Un grafo de estados consta de un conjunto de nodos que representan los estados del sistema conectados mediante flechas que representan las producciones que hacen pasar al sistema de un estado a otro. Dos nodos del grafo de estados estarán conectados mediante una flecha si y solo si existe una producción que transforma el sistema desde el estado situado en el origen de la flecha al estado situado en el extremo final de la flecha.

Debemos hacer hincapié en que al igual que el número de estados posibles nos impedía proporcionar de manera explícita soluciones preprogramadas para el puzzle de ocho piezas, ese mismo problema de magnitud nos impide representar explícitamente el grafo de estados completo. Un grafo de estados es por tanto simplemente una forma de conceptualizar el problema al que nos enfrentamos, pero no algo que podamos dibujar en su totalidad. De todos modos, es útil analizar (y posiblemente ampliar) parte del estado de grafos para el puzzle de ocho piezas que se muestra en la Figura 11.4.

Cuando se contempla en términos del grafo de estados, el problema al que se enfrenta el sistema de control se convierte en el de intentar encontrar una secuencia de flechas que conduzca desde el estado inicial al estado objetivo. Después de todo, esta secuencia de flechas representa una secuencia de producciones que resuelve el problema original. Por tanto, independientemente de la aplicación, la tarea del sistema de control puede considerarse como la de

Figura 11.4 Una pequeña parte del grafo de estados del puzzle de ocho piezas.



encontrar una ruta a través del grafo de estados. Esta visión universal de los sistemas de control es el premio que obtenemos al analizar los problemas que requieren razonamiento en términos de los sistemas de producción. Si un problema puede caracterizarse en términos de un sistema de producción, entonces su solución podrá formularse en términos de una búsqueda de una ruta.

Para hacer hincapié en este punto, consideremos cómo podríamos formular otras tareas en términos de sistemas de producción y cómo podríamos, por tanto, llevarlas a cabo en el contexto de sistemas de control que intenten encontrar rutas a través de grafos de estados. Uno de los problemas clásicos en el campo de la inteligencia artificial es el de practicar juegos tales como el ajedrez. Estos juegos tienen una complejidad moderada en un contexto bien definido, por lo que proporcionan un entorno ideal para la comprobación de teorías. En el ajedrez, los estados del sistema de producción subyacente son las posibles configuraciones del tablero; las producciones son los movimientos de las piezas y el sistema de control está integrado en los jugadores (humanos o no). El nodo inicial del grafo de estados representa el tablero con las piezas en sus posiciones iniciales. Ramificándose a partir de este nodo hay una serie de flechas que conducen a aquellas configuraciones del tablero que pueden alcanzarse después del primer movimiento de una partida; ramificándose a partir de esas configuraciones habrá otras flechas a aquellas configuraciones a las que se pueda llegar mediante el siguiente movimiento y así sucesivamente. Con esta formulación, podemos imaginar una partida de ajedrez compuesta por dos jugadores, cada uno de los cuales trata de encontrar una ruta a través de un enorme grafo de estados que les lleve hasta el nodo objetivo que hayan elegido.

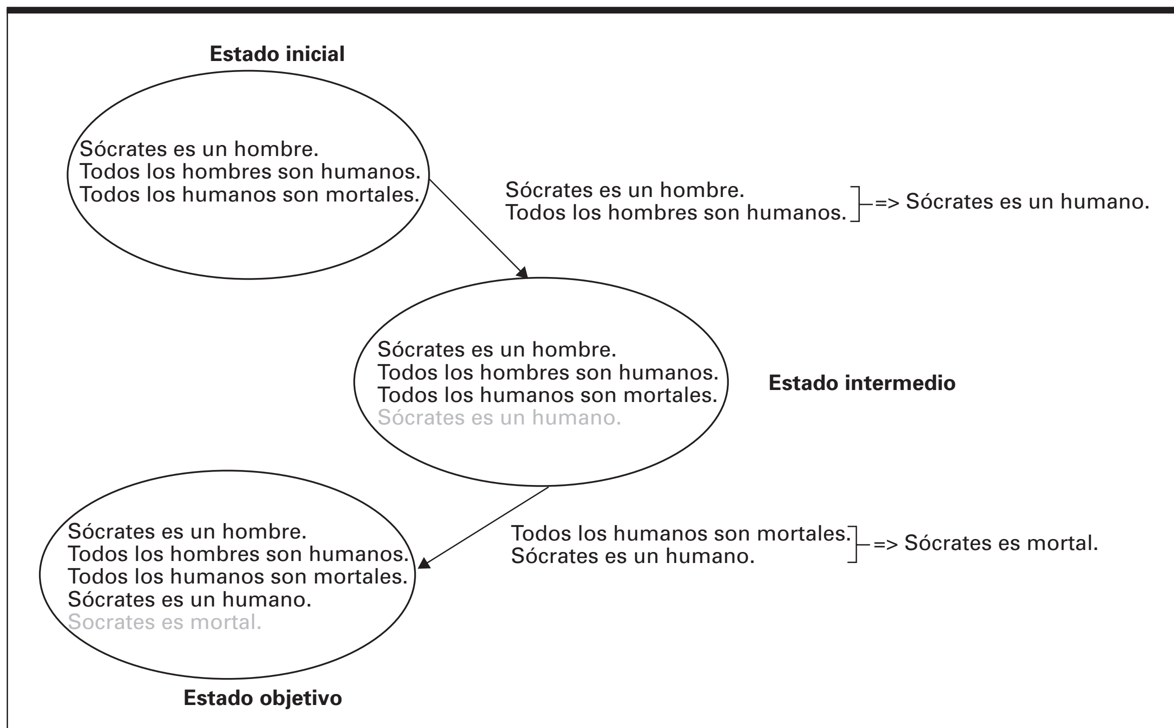
Quizá un ejemplo menos obvio de un sistema de producción sea el problema de obtener conclusiones lógicas a partir de una serie de hechos previa-

mente establecidos. Las producciones en este contexto serán las reglas de la lógica, denominadas **reglas de inferencia**, que permiten formar nuevos enunciados a partir de los antiguos. Por ejemplo, los enunciados "Todos los superhéroes son nobles" y "Superman es un superhéroe" pueden combinarse para producir el enunciado "Superman es noble". Los estados en este tipo de sistemas constan de conjuntos de enunciados que se sabe que son ciertos en determinados momentos concretos del proceso de deducción: el estado inicial será el conjunto de enunciados básicos (a menudo denominados axiomas) a partir de los cuales hay que extraer conclusiones y un estado objetivo será cualquier conjunto de enunciados que contenga la conclusión propuesta.

Por ejemplo, la Figura 11.5 muestra la parte de un grafo de estados que podría recorrerse para sacar la conclusión de que "Sócrates es mortal" a partir del conjunto de enunciados "Sócrates es un hombre", "Todos los hombres son humanos" y "Todos los humanos son mortales". Podemos ver en la figura cómo el cuerpo de conocimientos va pasando de un estado a otro, a medida que el proceso de razonamiento aplica las producciones apropiadas con el fin de generar enunciados adicionales.

Hoy día, esos sistemas de razonamiento que a menudo se implementan en lenguajes de programación lógica (Sección 6.7), constituyen la columna vertebral de la mayoría de los **sistemas expertos**, que son paquetes software diseñados para simular el razonamiento de tipo causa-efecto que los expertos humanos seguirían al verse enfrentados a las mismas situaciones. Por ejemplo, los sistemas expertos médicos se utilizan como ayuda para el diagnóstico de enfermedades o el desarrollo de tratamientos.

Figura 11.5 Razonamiento deductivo en el contexto de un sistema de producción.



Árboles de búsqueda

Hemos visto que en el contexto de un sistema de producción, el trabajo del sistema de control consiste en explorar el grafo de estados para encontrar una ruta que nos lleve desde el nodo inicial al nodo objetivo. Un método simple de realizar esta búsqueda consiste en recorrer cada una de las flechas que salen del estado inicial y anotar, en cada caso, el estado de destino. A continuación, se recorren las flechas que salen de esos nuevos estados y se anotan de nuevo esos resultados, y así sucesivamente. La búsqueda de un determinado objetivo se difundirá así a partir del estado inicial, como una gota de tinta en el agua. Este proceso continúa hasta que uno de los nuevos estados sea un estado objetivo, en cuyo punto sabremos que hemos encontrado una solución, por lo que el sistema de control tan solo necesitará encontrar las producciones que llevan desde el estado inicial hasta el estado objetivo a lo largo de la ruta descubierta.

El efecto de esta estrategia es el de construir un árbol, denominado **árbol de búsqueda**, que está compuesto por aquella parte del grafo de estados que ha sido explorada por el sistema de control. El nodo raíz del árbol de búsqueda será el estado inicial y los hijos de cada nodo son aquellos estados que resultan alcanzables a partir del padre aplicando una producción. Cada arco entre distintos nodos de un árbol de búsqueda representará la aplicación de una única producción y cada ruta que va desde la raíz a una hoja representa una ruta entre los correspondientes estados del grafo de estados.

En la Figura 11.7 se ilustra el árbol de búsqueda que se generaría al resolver el puzzle de ocho piezas a partir de la configuración mostrada en la Figura 11.6. La rama situada más a la izquierda de este árbol representa un intento de resolver el problema moviendo primero la pieza número 6 hacia arriba; la rama central representa el intento de solucionar el puzzle moviendo la pieza número 2 hacia la derecha y la rama de la derecha del árbol representa lo que se obtendría al mover la pieza número 5 hacia abajo. Como vemos, el árbol de búsqueda muestra que si comenzamos moviendo la pieza 6 hacia arriba, entonces la única producción permitida a continuación será mover la pieza 8 hacia la derecha. (De hecho, en ese punto también podríamos mover la pieza número 6 hacia abajo, pero eso sería invertir simplemente la producción anterior, volviendo a un estado por el que ya hemos pasado, lo que no es un movimiento muy inteligente.)

El estado objetivo aparece en el último nivel del árbol de búsqueda de la Figura 11.7. Puesto que esto indica que se ha encontrado una solución, el sistema de control puede terminar su procedimiento de búsqueda y comenzar a construir la secuencia de instrucciones que se utilizará para resolver el puzzle dentro del entorno externo. Para ello, basta con seguir el simple proceso de ascender por el árbol de búsqueda desde la posición del nodo objetivo al mismo

Figura 11.6 Un puzzle de ocho piezas sin resolver.

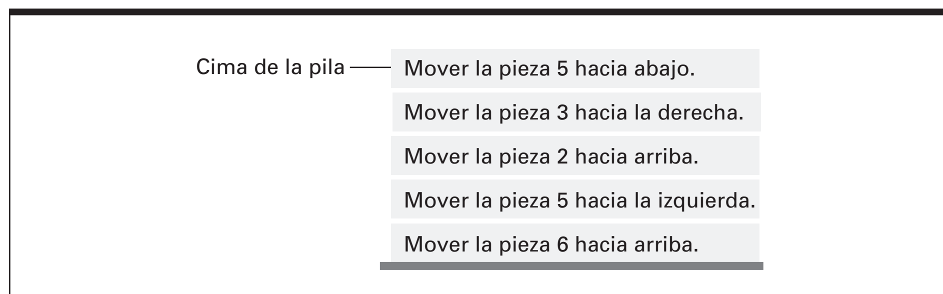
1	3	5
4	2	
7	8	6

tiempo que se insertan las producciones representadas por los arcos del árbol en una pila, a medida que nos las vamos encontrando. Aplicando esta técnica al árbol de búsqueda de la Figura 11.7 se obtiene la pila de producciones mostrada en la Figura 11.8. El sistema de control podrá ahora resolver el puzzle del mundo exterior ejecutando las instrucciones según las va sacando de esta pila.

Hay otra observación que conviene realizar. Recuerde que los árboles de los que hemos hablado en el Capítulo 8 utilizan un sistema de punteros que apunta hacia ***abajo*** del árbol, permitiéndonos así movernos desde un nodo

Figura 11.7 Un ejemplo de árbol de búsqueda.

Figura 11.8 Producciones apiladas para su ejecución posterior.



padre a sus hijos. Sin embargo, en el caso de un árbol de búsqueda, el sistema de control debe ser capaz de moverse desde un hijo hacia su padre, a medida que se desplaza hacia ***arriba*** del árbol, desde el estado objetivo hasta el estado inicial. Esos árboles se construyen con su sistema de punteros apuntando hacia arriba en lugar de hacia abajo. Es decir, cada nodo hijo contiene un puntero a su padre, en lugar de ser los nodos padre los que tienen punteros a los nodos hijo. (En algunas aplicaciones, se utilizan ambos conjuntos de punteros, con el fin de permitir el movimiento dentro del árbol en ambas direcciones.)

Heurística

Para nuestro ejemplo de la Figura 11.7, hemos elegido una configuración inicial que produce un árbol de búsqueda manejable. Pero en realidad, el árbol de búsqueda generado en un intento de resolver un problema más complejo podría crecer hasta alcanzar un tamaño muchísimo mayor. En una partida de ajedrez, hay veinte posibles movimientos para la primera jugada, así que el nodo raíz del árbol de búsqueda tendrá veinte hijos, en lugar de los tres que había en el caso de nuestro ejemplo. Además, una partida de ajedrez puede consistir fácilmente en 30 o 35 parejas de movimientos. Incluso en el caso del puzzle de ocho piezas, el árbol de búsqueda puede llegar a ser muy grande si no se alcanza el objetivo rápidamente. Como resultado, desarrollar un árbol de búsqueda completo puede ser tan imposible como representar el grafo de estados en su totalidad.

Una estrategia para resolver este problema consiste en cambiar el orden en el que se construye el árbol de búsqueda. En lugar de construirlo **en anchura** (lo que quiere decir que se construye nivel a nivel), podemos explorar las rutas más prometedoras con mayor profundidad y considerar las otras opciones solo si la selección original no conduce a ninguna parte. Esto da como resultado una construcción **en profundidad** del árbol de búsqueda, lo que quiere decir que el árbol se genera construyendo rutas verticales en lugar de niveles horizontales. Para ser más precisos, esta solución se denomina a menudo construcción **con prioridad para los mejores**, para reflejar el hecho de que la ruta vertical que se decide explorar es aquella que parece ofrecer el mayor potencial.

La solución de prioridad para los mejores es similar a la estrategia que utilizaríamos los humanos al enfrentarnos al puzzle de ocho piezas. Raramente analizaríamos varias opciones al mismo tiempo, que es lo que se hace en la solución de búsqueda en anchura. En lugar de ello, lo que probablemente haríamos sería seleccionar la opción que nos parezca más prometedora y seguirla en primer lugar. Observe que decimos la solución que ***parezca*** más prometedora. Rara vez sabremos con seguridad qué opción es la mejor en un punto concreto. Simplemente seguimos nuestra intuición, que por supuesto puede no llevarnos a ninguna parte. Pero aunque en ocasiones no nos lleve a ninguna parte, el uso de esa información intuitiva parece proporcionar a los seres humanos una ventaja con respecto a los métodos de fuerza bruta consistentes en dedicar la misma atención a todas las opciones posibles. Por tanto, parece prudente aplicar esos métodos intuitivos a los sistemas de control automatizados.

Para ello, necesitamos una forma de identificar cuáles son los estados más prometedores de entre un conjunto de estados posibles. Nuestra solución consiste en utilizar un **heurístico**, que en nuestro caso es un valor cuantitativo

asociado con cada estado; con ese valor se intenta medir la "distancia" que hay entre ese estado y el objetivo más próximo. En cierto sentido, nuestro heurístico es una medida del coste previsto. Dada la posibilidad de elegir entre dos estados, aquel que tenga el valor heurístico más bajo será el estado a partir del cual se podrá, aparentemente, alcanzar un objetivo con un menor coste. Este estado representará, por tanto, la dirección que debemos explorar.

Un heurístico debe tener dos características. En primer lugar, debe ser una estimación razonable de la cantidad de trabajo restante en la solución si se llegase a alcanzar el estado asociado. Esto quiere decir que puede proporcionar información significativa a la hora de elegir entre varias opciones: cuanto mejor sea la estimación proporcionada por el heurístico, mejores serán las decisiones que tomemos basándonos en esa información. En segundo lugar, el heurístico debe ser fácil de calcular. Esto quiere decir que su uso debe darnos la posibilidad de beneficiar al proceso de búsqueda, en lugar de entorpecerlo. Si el cálculo del heurístico es extremadamente complicado, entonces a lo mejor tiene más sentido que invirtamos nuestro tiempo realizando una búsqueda en anchura.

Un heurístico simple en el caso del puzzle de ocho piezas consistiría en estimar la "distancia" hasta el objetivo contando el número de piezas que están descolocadas, esta solución se basaría en la conjetura de que un estado en el que haya cuatro piezas descolocadas está más lejos del objetivo (y es por tanto menos prometedor) que un estado en el que solo estén descolocadas dos piezas. Sin embargo, este heurístico no tiene en cuenta lo lejos que las piezas mal colocadas están de su posición correcta. Si las dos piezas del segundo caso están

Inteligencia basada en el comportamiento

Los primeros trabajos en el campo de la inteligencia artificial enfocaban el problema en el contexto de tratar de escribir explícitamente programas que simularan la inteligencia. Sin embargo, muchos expertos argumentan hoy día que la inteligencia del ser humano no está basada en la ejecución de programas complejos, sino en una serie de funciones simples de tipo estímulo-respuesta, que han ido evolucionando a lo largo de las generaciones. Esta teoría de la "inteligencia" se conoce con el nombre de inteligencia basada en el comportamiento, porque las funciones "inteligentes" de tipo estímulo-respuesta parecen ser el resultado de comportamientos que han hecho que ciertos individuos sobrevivan y se reproduzcan, mientras que otros no pudieron hacerlo.

La inteligencia basada en el comportamiento parece responder a diversas cuestiones que se plantean en el campo de la inteligencia artificial, como por ejemplo por qué las máquinas basadas en la arquitectura de von Neumann tienen unas habilidades computacionales muy superiores a las de los humanos pero sin embargo luchan infructuosamente por exhibir sentido común. Por ello, la inteligencia basada en el comportamiento promete ser una de las influencias principales en las investigaciones realizadas dentro del campo de la inteligencia artificial. Como se describe en el texto, las técnicas basadas en el comportamiento se han aplicado en el campo de las redes neuronales artificiales con el fin de enseñar a las neuronas a comportarse de determinadas maneras deseadas; se ha usado también en el campo de los algoritmos genéticos para proporcionar una alternativa al proceso de programación más tradicional y en robótica para mejorar el comportamiento de las máquinas mediante estrategias reactivas.

muy alejadas de sus posiciones correctas, podrían necesitarse muchas producciones para moverlas a través del puzzle.

Por tanto, un heurístico ligeramente mejor sería medir la distancia a la que está cada pieza de su destino, y sumar esos valores para obtener un único valor global. Una pieza que esté inmediatamente adyacente a su destino final se asociaría con una distancia de uno, mientras que una pieza cuya esquina toque el cuadrado de su destino final se asociaría con una distancia de dos (porque debe moverse al menos una posición en sentido vertical y una posición en sentido horizontal). Este heurístico es fácil de calcular y produce una estimación burda del número de movimientos necesario para transformar el puzzle desde su estado actual hasta el objetivo. Por ejemplo, el valor heurístico asociado con la configuración de la Figura 11.9 es siete (porque las piezas 2, 5 y 8 están a una distancia de uno de su destino final, mientras que las piezas 3 y 6 están a una distancia de dos de su posición correcta). De hecho, en realidad hacen falta siete movimientos para transformar esta configuración del puzzle en la configuración correcta.

Ahora que disponemos de un heurístico para el puzzle de ocho piezas, el siguiente paso consiste en incorporarlo a nuestro proceso de toma de decisiones. Recuerde que un ser humano enfrentado a una decisión tiende a seleccionar la opción que parece más próxima al objetivo. Por tanto, nuestro procedimiento de búsqueda debería considerar el heurístico de cada nodo hoja del árbol de búsqueda y continuar la búsqueda de un nodo hoja que tenga asociado el valor más pequeño. Esta estrategia es la adoptada en la Figura 11.10, que presenta un algoritmo para desarrollar un árbol de búsqueda y ejecutar la solución obtenida.

Apliquemos este algoritmo al puzzle de ocho piezas partiendo de la configuración inicial mostrada en la Figura 11.6. En primer lugar, establecemos este estado inicial como el nodo raíz y anotamos su valor heurístico, que es cinco. Después, la primera pasada a través del cuerpo de la sentencia while nos dice que añadamos los tres nodos que pueden alcanzarse a partir del estado inicial, como puede verse en la Figura 11.11. Observe que hemos anotado el valor heurístico de cada nodo hoja entre paréntesis debajo del nodo.

El nodo objetivo no ha sido alcanzado por lo que volvemos a pasar por el cuerpo de la sentencia while, ampliando esta vez nuestra búsqueda a partir del nodo situado más a la izquierda (“el nodo hoja situado más a la izquierda que tenga el valor heurístico más bajo”). Después de esto, el árbol de búsqueda tendrá la forma mostrada en la Figura 11.12.

Figura 11.9 Un puzzle de ocho piezas sin resolver.

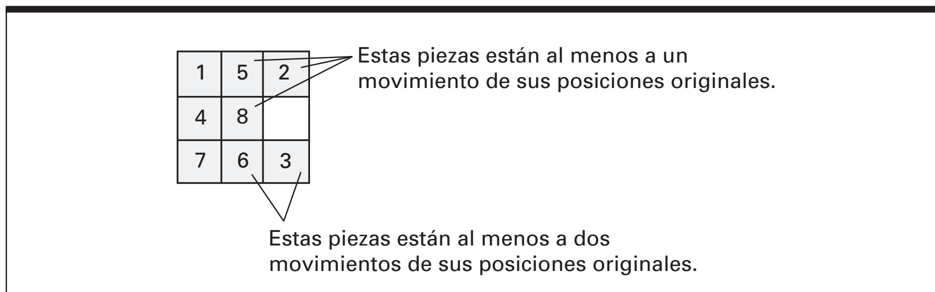


Figura 11.10 Un algoritmo para un sistema de control que utiliza heurísticos.

```
Establecer el nodo inicial del grafo de estados como el nodo raíz
    del árbol de búsqueda y anotar su valor heurístico.
while (el nodo objetivo no se haya alcanzado) do
    [Seleccionar el nodo hoja de más a la izquierda que tenga el
        valor heurístico más bajo de todos los nodos hoja.
     Asociar a este nodo seleccionado como hijos aquellos nodos
        que puedan alcanzarse mediante una única producción.
     Anotar el heurístico de cada uno de estos nuevos nodos dentro
        del árbol de búsqueda a continuación del nodo
    ]
Recorrer el árbol de búsqueda desde el nodo objetivo hasta la raíz,
    introduciendo en una pila la producción asociada con cada
    arco recorrido.
Resolver el problema original ejecutando las producciones según
    se van sacando de la pila.
```

El valor heurístico del nodo hoja más a la izquierda ahora es cinco, lo que indica que quizá esta rama no sea después de todo una buena opción para continuar el análisis. El algoritmo tiene esto en cuenta y en la siguiente pasada a través de la sentencia while decide que hay que ampliar el árbol a partir del nodo situado más a la derecha (que ahora es el "el nodo hoja situado más a la izquierda que tenga el valor heurístico más bajo"). Habiéndolo ampliado de esta forma, el árbol de búsqueda tendrá la apariencia que se muestra en la Figura 11.13.

En este punto, el algoritmo parece estar bien encaminado. Puesto que el valor heurístico de este último nodo es solo tres, la sentencia while dice que hay que continuar explorando esta ruta y la búsqueda prosigue hacia el objetivo, generando el árbol de búsqueda que se ilustra en la Figura 11.14. Comparando esto con el árbol de la Figura 11.7 vemos que, incluso con el movimiento en falso que realizó al principio el nuevo algoritmo, el uso de la información heurística ha reducido considerablemente el tamaño del árbol de búsqueda y ha generado un proceso mucho más eficiente.

Figura 11.11 Comienzo de nuestra búsqueda heurística.

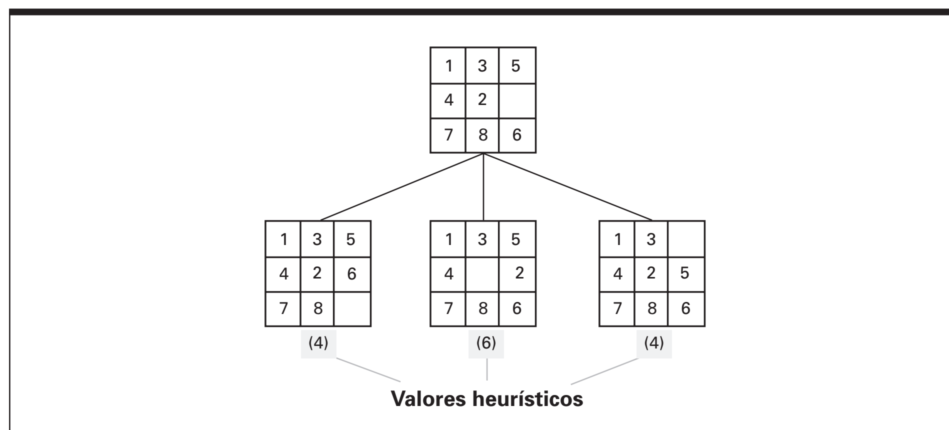
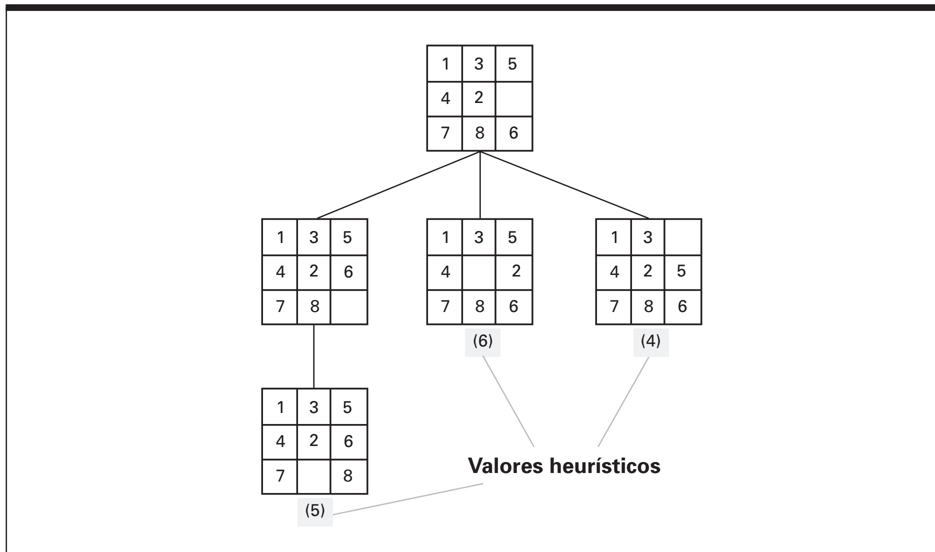


Figura 11.12 El árbol de búsqueda después de dos pasadas.



Después de alcanzar el estado objetivo, la sentencia while termina y el algoritmo continúa recorriendo el árbol desde el nodo objetivo hasta la raíz, insertando en una pila las producciones que se va encontrando. La pila resultante tendrá el aspecto que hemos mostrado anteriormente en la Figura 11.8.

Finalmente, el algoritmo ejecuta esas producciones a medida que las va sacando de la pila. Llegados a este punto, observaríamos cómo la máquina solucionadora de puzzles hace descender su dedo actuador y comienza a mover las piezas.

Figura 11.13 El árbol de búsqueda después de tres pasadas.

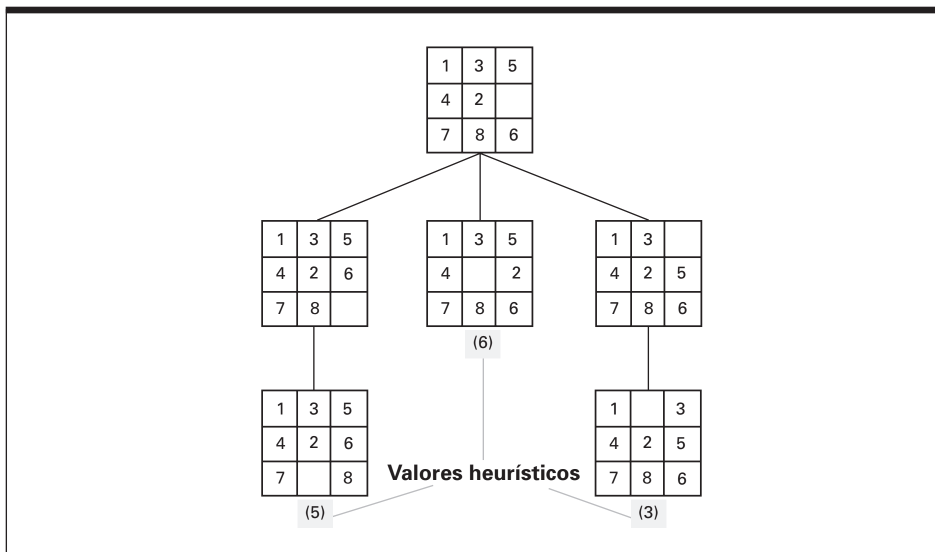
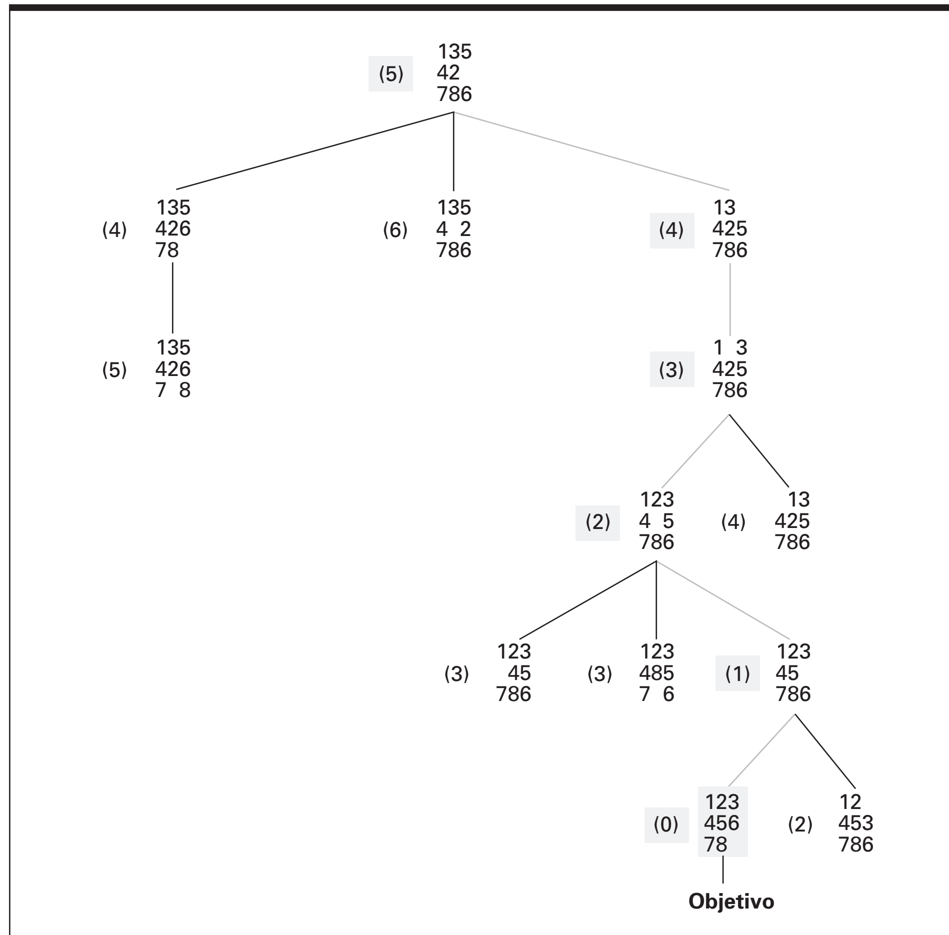


Figura 11.14 El árbol de búsqueda completo formado por nuestro sistema heurístico.



Conviene hacer un último comentario en relación con la búsqueda heurística. El algoritmo que hemos propuesto en esta sección, que a menudo se denomina algoritmo de búsqueda por elección del mejor (*best-fit search*), no garantiza poder encontrar la mejor solución en todas las aplicaciones. Por ejemplo, a la hora de buscar una ruta hasta una ciudad utilizando un Sistema de posicionamiento global (GPS, ***Global Positioning System***) en un automóvil, lo que querríamos es encontrar la ruta más corta y no cualquier ruta. El **algoritmo A*** (léase algoritmo A asterisco) es una versión modificada de nuestro algoritmo de búsqueda por elección del mejor que permite encontrar una solución óptima. La principal diferencia entre los dos algoritmos es que, además de un valor heurístico, el algoritmo A* tiene en consideración el "coste acumulado" en el que se incurre para alcanzar cada nodo hoja, antes de seleccionar el siguiente nodo que hay que expandir (en el caso del GPS de un automóvil, este coste es la distancia recorrida, que el GPS obtiene de su base de datos interna). Por tanto, el algoritmo A* basa sus decisiones en estimaciones del coste requerido para completar cada ruta potencial, en lugar de basarlas tan solo en estimaciones de los restantes costes.

Cuestiones y ejercicios

1. ¿Qué importancia tienen los sistemas de producción en el campo de la inteligencia artificial?
2. Dibuje una parte del grafo de estados del puzzle de ocho piezas que rodee al nodo representado por el siguiente estado:

4	1	3
	2	6
7	5	8

3. Utilizando la solución de búsqueda en anchura, dibuje el árbol de búsqueda que un sistema de control construiría a la hora de resolver el puzzle de ocho piezas correspondiente al siguiente estado inicial:

1	2	3
4	8	5
7	6	

4. Utilice lápiz, papel y la solución de búsqueda en anchura para tratar de construir el árbol de búsqueda que se generará al resolver el puzzle de ocho piezas a partir del siguiente estado inicial. (No es necesario que termine el problema.) ¿Con qué problemas se ha encontrado?

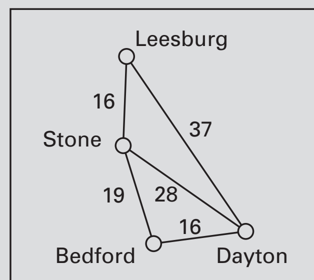
4	3	
2	1	8
7	6	5

5. ¿Qué analogía podría establecerse entre nuestro sistema heurístico para la resolución del puzzle de ocho piezas y un montañero que tratara de alcanzar el pico de una montaña considerando únicamente el terreno local y siguiendo siempre en la dirección con la pendiente más pronunciada?
6. Utilizando el heurístico presentado en esta sección, aplique el algoritmo de búsqueda por elección del mejor de la Figura 11.10 al problema de resolver el siguiente puzzle de ocho piezas:

1	2	3
4		8
7	6	5

7. Refine nuestro método de cálculo del valor heurístico para el estado del puzzle de ocho piezas, de modo que el algoritmo de búsqueda de la Figura 11.10 no tome la decisión incorrecta, como hacía en el ejemplo de esta sección. ¿Puede encontrar un ejemplo en el que su heurístico siga haciendo que la búsqueda recorra un camino equivocado?

8. Utilice el árbol de búsqueda generado mediante el algoritmo de búsqueda por elección del mejor (Figura 11.10) para calcular la ruta entre las ciudades de Leesburg y Bedford. Cada nodo del árbol de búsqueda será una ciudad en el mapa. Comience con un nodo para Leesburg. A la hora de expandir un nodo, añada solamente las ciudades que estén directamente conectadas con la ciudad que esté expandiendo. Anote en cada nodo la distancia en línea recta hasta Bedford y utilice ese valor como valor heurístico. ¿Cuál es la solución encontrada por el algoritmo de búsqueda por elección del mejor? ¿Se corresponde esa solución encontrada con la ruta más corta?



Distancia en línea recta a Bedford desde

Dayton	16
Leesburg	34
Stone	19

9. El algoritmo A* modifica el algoritmo de búsqueda por elección del mejor de dos formas significativas. En primer lugar, anota el coste real necesario para alcanzar cada estado. En el caso de una ruta en un mapa, el coste real es la distancia recorrida. En segundo lugar, a la hora de seleccionar el nodo que hay que expandir, elige el nodo que tenga un valor mínimo de la suma entre el coste real y el valor heurístico. Dibuje el árbol de búsqueda de la Cuestión 8 que resultaría al efectuar estas dos modificaciones. Anote en cada nodo la distancia recorrida hasta la ciudad, el valor heurístico para alcanzar el objetivo y su suma. ¿Cuál es la solución encontrada? ¿Se corresponde la solución encontrada con la ruta más corta?

11.4 Áreas adicionales de investigación

En esta sección, vamos a explorar las cuestiones relacionadas con el manejo del conocimiento, el aprendizaje y la manera de abordar problemas muy complejos, cuestiones todas ellas que continúan planteando desafíos a los investigadores en el campo de la inteligencia artificial. Estas actividades implican capacidades que parecen sencillas para la mente humana, pero que son aparentemente muy complicadas para las máquinas. Por ahora, buena parte del progreso obtenido a la hora de desarrollar agentes "inteligentes" se ha conseguido, básicamente, evitando la confrontación directa con estos problemas, por ejemplo aplicando atajos ingeniosos o limitando el ámbito de aplicación de un problema.

Representación y manipulación del conocimiento

En las explicaciones acerca de la percepción vimos que la comprensión de imágenes requiere una cantidad significativa de conocimientos acerca de los elementos de la imagen y que el significado de una frase puede depender de su

contexto. Estos son ejemplos del papel que desempeña el almacén de conocimientos, que a menudo se denomina **conocimientos del mundo real**, mantenido por la mente humana. De alguna forma, las personas almacenamos cantidades masivas de información y sabemos utilizar esa información con una notable eficiencia. Proporcionar a las máquinas esta capacidad es uno de los desafíos principales en el campo de la inteligencia artificial.

El objetivo subyacente es encontrar formas de representar y almacenar el conocimiento. Esto se ve complicado por el hecho de que, como ya hemos visto, el conocimiento puede presentarse tanto de forma declarativa como procedimental. Por ello, representar el conocimiento no consiste simplemente en la representación de hechos, sino que abarca un espectro mucho más amplio. Es por tanto cuestionable la tesis de que pueda llegar a encontrarse un único esquema para representar todas las formas de conocimiento.

Sin embargo, el problema no consiste simplemente en almacenar y representar conocimiento. También se debe poder acceder fácilmente al conocimiento almacenado y conseguir esta accesibilidad es un auténtico desafío. Las redes semánticas, tal como las hemos presentado en la Sección 11.2, suelen utilizarse como medio de representación y almacenamiento del conocimiento, pero extraer información de ellas puede ser problemático. Por ejemplo, la importancia de la frase "María golpeó a Juan" depende de las edades relativas de María y Juan (¿son esas edades 2 y 30 años, o viceversa?). Esta información estaría almacenada en la red semántica completa sugerida en la Figura 11.3, pero extraer dicha información durante el análisis contextual podría requerir una tarea significativa de búsqueda a través de la red.

Otro problema más que también está relacionado con el acceso al conocimiento es el de identificar aquellos conocimientos que se relacionan implícitamente, en lugar de explícitamente, con la tarea que tengamos entre manos. En lugar de responder a la cuestión "¿Ganó Arturo la carrera?" con un seco "No", lo que queremos es disponer de un sistema que pueda responder diciendo "No, se puso enfermo y no pudo competir". En la siguiente sección exploraremos el concepto de memoria asociativa, que es una de las áreas de investigación que está intentando resolver este problema de la información relacionada. Sin embargo, la tarea no consiste simplemente en extraer información relacionada; necesitamos sistemas que puedan distinguir entre información relacionada e información relevante. Por ejemplo, una respuesta como "No, nació en enero y su hermana se llama Elisa" no se consideraría una respuesta adecuada para la pregunta anterior, aún cuando la información comunicada esté relacionada en cierta manera.

Otra técnica para desarrollar sistemas mejores de extracción del conocimiento consiste en insertar diversos tipos de razonamiento dentro del proceso de extracción, lo que da como resultado una técnica que se denomina **meta-razonamiento**, lo que quiere decir razonamiento acerca del razonamiento. Un ejemplo, originalmente utilizado en el contexto de las búsquedas de bases de datos, consiste en aplicar la **suposición del mundo cerrado**, que es la suposición de que un enunciado es falso a menos que pueda derivarse explícitamente de la información disponible. Por ejemplo, es esa suposición del mundo cerrado la que permite a una base de datos concluir que Juan García no está suscrito a ninguna revista concreta, aún cuando la base de datos no contenga ninguna información en absoluto acerca de Juan. El proceso consiste en observar que

Juan García no se encuentra en la lista de subscriptores y luego aplicar la suposición del mundo cerrado para concluir que Juan García no es un subscriptor.

Si se analiza superficialmente, la suposición del mundo cerrado parece trivial, pero tiene consecuencias que ilustran cómo algunas técnicas de metarazonamiento aparentemente inocentes pueden tener efectos sutiles indeseados. Por ejemplo, suponga que el único conocimiento que tenemos es el enunciado

Mickey es un ratón OR Donald es un pato.

A partir de este enunciado aislado no podemos concluir que Mickey sea realmente un ratón. Por tanto, la suposición del mundo cerrado nos fuerza a concluir que el enunciado

Mickey es un ratón.

es falso. De manera similar, la suposición del mundo cerrado nos fuerza a concluir que el enunciado

Donald es un pato.

es falso. Por tanto, la suposición del mundo cerrado nos ha conducido a la conclusión contradictoria de que, aunque al menos uno de los enunciados debe ser cierto, ambos son falsos. Entender las consecuencias de estas técnicas de metarazonamiento de apariencia inocente es un objetivo de las actuales investigaciones tanto en el campo de la inteligencia artificial como en el de las bases de datos, y resalta algunas de las complejidades inherentes al desarrollo de sistemas inteligentes.

Finalmente, existe el problema conocido con el nombre de **problema del marco**, de mantener actualizado el conocimiento almacenado dentro de un entorno cambiante. Si un agente inteligente va a utilizar sus conocimientos con el fin de determinar su comportamiento, entonces esos conocimientos deben estar actualizados. Pero la cantidad de conocimiento requerido para permitir un comportamiento inteligente puede ser enorme y mantener ese conocimiento en un entorno cambiante puede ser una tarea tremendamente compleja. Un factor que viene a complicar aún más el panorama es que los cambios dentro de un entorno suelen alterar indirectamente otros elementos de información y tener en cuenta esas consecuencias indirectas es difícil. Por ejemplo, si un jarrón se cae y se rompe, nuestro conocimiento de la situación ya no contendrá el hecho de que hay agua en el jarrón, aún cuando el vertido del agua solo está relacionado indirectamente con la ruptura del jarrón. Por tanto, resolver el problema del marco no solo requiere la capacidad de almacenar y extraer cantidades masivas de información de una manera eficiente, sino que también exige que el sistema de almacenamiento reaccione apropiadamente a las consecuencias indirectas.

Aprendizaje

Además de representar y manipular el conocimiento, nos gustaría proporcionar a los agentes inteligentes la capacidad de adquirir nuevos conocimientos. Siempre podemos "enseñar" a un agente basado en computadora escribiendo e instalando un nuevo programa o almacenando nuevas informaciones en su conjunto de datos almacenados, pero lo que nos gustaría es que los agentes

inteligentes fueran capaces de aprender por su cuenta. Queremos disponer de agentes que se adapten a entornos cambiantes y que realicen tareas para las que no podemos escribir fácilmente programas de antemano. Un robot diseñado para tareas domésticas se encontrará con nuevos muebles, nuevos electrodomésticos, nuevas mascotas e incluso nuevos propietarios. Un vehículo autónomo que se conduzca solo deberá adaptarse a las variaciones en las líneas que marcan el perfil de las carreteras. Los agentes que forman parte en un juego deben ser capaces de desarrollar y aplicar nuevas estrategias.

Una forma de clasificar las soluciones para el problema de aprendizaje en una computadora es según el nivel de intervención humana requerido. En el primer nivel se encontraría el aprendizaje por **imitación**, en el que una persona demuestra directamente los pasos que componen una tarea (quizá llevando a cabo una secuencia de operaciones en la computadora o guiando físicamente a un robot a través de una secuencia de movimientos) y la computadora se limita a anotar los pasos. Esta forma de aprendizaje se ha utilizado a lo largo de los años en programas de aplicación tales como las hojas de cálculo y los procesadores de texto, en los que pueden guardarse las secuencias de comandos más frecuentemente utilizadas y posteriormente reproducirlas mediante una única orden. Observe que el aprendizaje por imitación asigna muy poca responsabilidad al agente.

En el siguiente nivel se encuentra el aprendizaje mediante **entrenamiento supervisado**. En el entrenamiento supervisado, una persona identifica la respuesta correcta para una serie de ejemplos y luego el agente generaliza a partir de esos ejemplos, con el fin de desarrollar un algoritmo que se pueda aplicar a nuevos casos. La serie de ejemplos se denomina **conjunto de entrenamiento**. Entre las aplicaciones típicas del entrenamiento supervisado se incluyen el aprendizaje para reconocer la escritura manuscrita o la voz de una persona, el aprendizaje para distinguir entre correo electrónico basura y deseado y el aprendizaje para identificar una enfermedad a partir de un conjunto de síntomas.

Un tercer nivel sería el aprendizaje por **refuerzo**. En este tipo de aprendizaje, al agente se le proporciona una regla general para que juzgue por sí mismo si ha tenido éxito o ha fallado a la hora de realizar una tarea, durante un proceso de prueba y error. El aprendizaje por refuerzo es adecuado para aprender a jugar al ajedrez o las damas, donde el éxito o el fracaso son fáciles de definir. A diferencia del entrenamiento supervisado, el aprendizaje por refuerzo permite al agente actuar autónomamente a medida que aprende a mejorar su comportamiento a lo largo del tiempo.

El aprendizaje continúa siendo un campo de investigación lleno de desafíos, ya que no se ha encontrado ningún principio general de carácter universal que cubra todas las posibles actividades de aprendizaje. Sin embargo, existen numerosos ejemplos de los progresos realizados. Uno de ellos es ALVINN (*Autonomous Land Vehicle in a Neural Net*, Vehículo terrestre autónomo en una red neuronal), un sistema desarrollado en la Universidad Carnegie Mellon para aprender a conducir un camión con una computadora de abordo utilizando una videocámara como entrada. La técnica empleada era el entrenamiento supervisado. ALVINN recopilaba datos de un conductor humano y utilizaba esos datos para ajustar sus propias decisiones de conducción. A medida que aprendía, predecía qué acciones realizar, comprobaba su predic-

El conocimiento en la programación lógica

Una preocupación importante a la hora de representar y almacenar conocimiento es que hay que llevar a cabo esa tarea de una forma que sea compatible con el sistema que luego vaya a acceder a ese conocimiento. Es por esta razón por lo que la programación lógica (véase la Sección 6.7) suele resultar ventajosa. En ese tipo de sistemas, el conocimiento está representado por enunciados "lógicos" como

Dumbo es un elefante.

y

X es un elefante implica que X es gris.

Dichos enunciados pueden representarse mediante sistemas de notación que están muy adaptados a la aplicación de reglas de inferencia. A su vez, las secuencias de razonamiento deductivo, como la que vimos en la Figura 11.5, pueden implementarse de manera muy sencilla. Por tanto, en la programación lógica, la representación y el almacenamiento de conocimiento están bien integrados con el proceso de extracción y aplicación del conocimiento. Podríamos decir que los sistemas de programación lógica proporcionan una interfaz "adaptada" entre el conocimiento almacenado y su aplicación.

ción con los datos obtenidos del conductor humano y luego modificaba sus parámetros para aproximarse más a las decisiones de conducción tomadas por esa persona. ALVINN tuvo tanto éxito que podía conducir el camión a cien kilómetros por hora, lo que condujo a investigaciones adicionales que han terminado produciendo sistemas de control que han sido capaces de conducir en una autopista llena de tráfico a las velocidades apropiadas.

Por último, es necesario realizar algunos comentarios acerca de un fenómeno estrechamente relacionado con el aprendizaje: nos referimos al descubrimiento. La distinción entre estos dos conceptos es que el aprendizaje está "basado en objetivos", mientras que el descubrimiento no lo está. El término ***descubrimiento*** tiene una connotación de algo inesperado que no está presente en el concepto de aprendizaje. Podemos disponernos a aprender un idioma extranjero o a conducir un vehículo, pero podemos descubrir que esas tareas son más difíciles de lo esperado. Un explorador puede descubrir un gran lago, aún cuando el objetivo fuera simplemente aprender qué es lo que había en esa región.

Desarrollar agentes con la capacidad de realizar descubrimiento de manera eficiente requiere que el agente sea capaz de identificar "cadenas de pensamientos" potencialmente fructíferas. Aquí, la habilidad del descubrimiento depende enormemente de la capacidad de razonar y del uso de heurísticos. Además, muchas aplicaciones potenciales de la capacidad de descubrimiento requieren que un agente sea capaz de distinguir resultados significativos de otros que sean insignificantes. Un agente de minería de datos, por ejemplo, no debe informar de toda relación trivial que encuentre.

Como ejemplos de éxito a la hora de desarrollar sistemas de descubrimiento basados en computadora podemos citar a Bacon, denominado así en honor del filósofo Sir Francis Bacon, que ha descubierto (o quizá deberíamos decir "redescubierto") la ley de Ohm de la electricidad, la tercera ley de Kepler

del movimiento de los planetas y el principio de conservación de la cantidad de movimiento. Quizá más persuasivo sea el sistema AUTOCLASS que, utilizando datos espectrales infrarrojos ha descubierto nuevas clases de estrellas que antes eran desconocidas en el campo de la astronomía, lo que constituye un verdadero descubrimiento científico efectuado por una computadora.

Algoritmos genéticos

El algoritmo A* (presentado en la sección anterior) permite encontrar la solución óptima a muchos problemas de búsqueda; sin embargo, existen muchos problemas que son demasiado complejos para poderlos resolver mediante esas técnicas de búsqueda (la ejecución sobrepasa la memoria disponible o no puede completarse dentro de un periodo de tiempo razonable). Para este tipo de problemas, en ocasiones puede descubrirse una solución mediante un proceso evolutivo que implica múltiples generaciones de soluciones de prueba. Esta estrategia es la base de lo que se denomina **algoritmos genéticos**. En esencia, los algoritmos genéticos descubren una solución combinando un comportamiento aleatorio con una simulación de la teoría reproductiva y del proceso evolutivo de la selección natural.

Un algoritmo genético comienza generando un conjunto aleatorio de soluciones de prueba. Cada solución se selecciona al azar. (En el caso del puzzle de ocho piezas, una solución de prueba podría ser una secuencia aleatoria del movimiento de las piezas.) Cada solución de prueba se denomina **cromosoma** y cada componente del cromosoma se llama **gen** (en el caso del puzzle de ocho piezas un gen sería un único movimiento de una pieza).

Puesto que los cromosomas iniciales se han seleccionado aleatoriamente, es bastante poco probable que ninguno de ellos represente una solución al problema que intentamos resolver. Por tanto, el algoritmo genético pasa a generar un nuevo conjunto de cromosomas, en el que cada cromosoma es un descendiente (hijo) de dos cromosomas (padres) del conjunto anterior. Los padres se seleccionan aleatoriamente dentro del conjunto, proporcionándose una preferencia probabilística a aquellos cromosomas que parezcan proporcionar las mejores oportunidades de conducir a una solución, con lo que se simula el principio evolutivo de la supervivencia del más adaptado. (Determinar qué cromosomas son los mejores candidatos para reproducirse es quizá el paso más problemático dentro del proceso de un algoritmo genético.) Cada hijo se forma mediante la combinación aleatoria de los genes de los padres. Además, un hijo puede ocasionalmente sufrir una mutación de carácter aleatorio (por ejemplo, intercambiar dos movimientos). La esperanza es que al repetir este proceso una y otra vez vayan evolucionando soluciones de prueba cada vez mejores, hasta descubrir una que sea muy buena, si es que no es la mejor posible. Lamentablemente, no existe ninguna garantía de que el algoritmo genético termine por encontrar una solución, a pesar de lo cual las investigaciones han demostrado que los algoritmos genéticos pueden resultar efectivos a la hora de resolver un rango sorprendentemente amplio de problemas complejos.

Cuando se aplica a la tarea del desarrollo de programas, la solución mediante algoritmo genético se conoce con el nombre de **programación evolutiva**. En este caso, el objetivo es desarrollar programas permitiendo que estos evolucionen, en lugar de escribirlos explícitamente. Los investigadores han

aplicado técnicas de programación evolutiva al proceso de desarrollo de programas utilizando lenguajes de programación funcional. La solución adoptada consiste en comenzar con un conjunto de programas que contengan una amplia variedad de funciones. Las funciones de este conjunto inicial forman el "conjunto de genes" a partir de los cuales se construirán las generaciones futuras de programas. Después, se permite que el proceso evolutivo funcione durante muchas generaciones, esperando que al producir cada generación a partir de los programas que mejor se han comportado en la generación anterior, irá evolucionando hacia una solución para nuestro problema objetivo.

Cuestiones y ejercicios

1. ¿Qué queremos decir con la frase ***conocimiento del mundo real*** y cuál es su importancia en el campo de la inteligencia artificial?
2. Una base de datos de subscriptores a revistas suele contener una lista de los subscriptores de cada revista pero no contiene una lista de personas que no están suscritas. ¿Cómo puede entonces una de esas bases de datos determinar que una persona no está suscrita a una determinada revista?
3. Resuma el denominado problema del marco.
4. Identifique tres formas de entrenar a una computadora. ¿Cuál de ellas no requiere una intervención humana directa?
5. ¿En qué difieren las técnicas evolutivas de las técnicas más tradicionales de resolución de problemas?

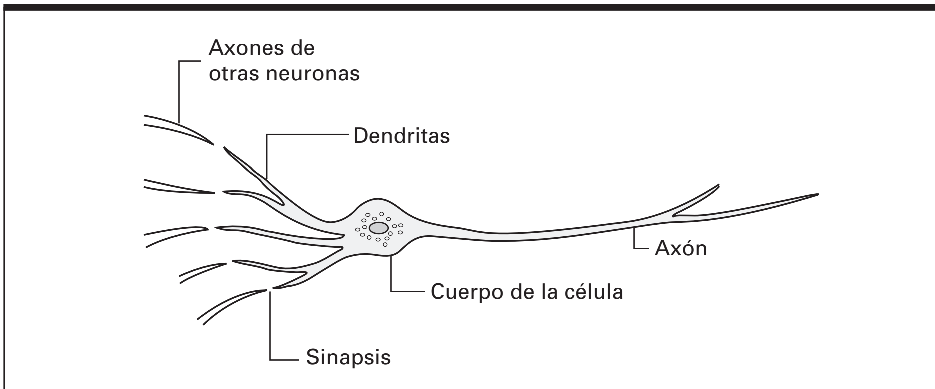
11.5 Redes neuronales artificiales

A pesar de todos los progresos realizados en el campo de la inteligencia artificial, muchos problemas continúan desafiando las habilidades de las computadoras que utilizan soluciones algorítmicas tradicionales. Las secuencias de instrucciones no parecen ser capaces de percibir y razonar a niveles comparables con los de la mente humana. Por esta razón, muchos investigadores están volviendo su atención hacia soluciones que tratan de aprovechar fenómenos observados en la naturaleza. Una de esas técnicas son los algoritmos genéticos presentados en la sección anterior y otra es la utilización de redes neuronales artificiales.

Propiedades básicas

Las redes neuronales artificiales proporcionan un modelo de procesamiento por computadora que simula las redes de neuronas de los sistemas biológicos de los seres vivos. Una neurona biológica es una única celda con una serie de tentáculos de entrada denominadas dendritas y un tentáculo de salida denominado axón (Figura 11.15). Las señales transmitidas a través del axón de una célula reflejan si la célula se encuentra en un estado inhibido o excitado. Este estado está determinado por la combinación de las señales recibidas a través de

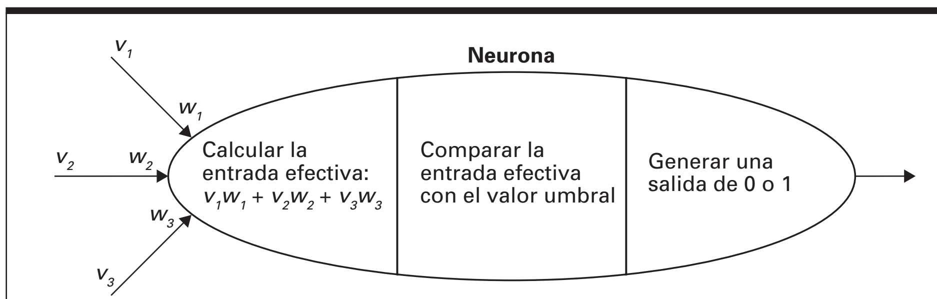
Figura 11.15 Una neurona de un sistema biológico de un ser vivo.



las dendritas de la célula. Estas dendritas captan señales procedentes de los axones de otras células a través de pequeños huecos denominados sinapsis. Las investigaciones sugieren que la conductividad a través de una sinapsis está controlada por la composición química de la sinapsis. Es decir, el que la señal de entrada concreta tenga un efecto excitador o inhibidor sobre la neurona está determinado por la composición química de la sinapsis. Así, se cree que una red neuronal biológica aprende ajustando estas conexiones químicas entre neuronas.

Una neurona en una red neuronal artificial es una unidad software que imita este modelo básico de una neurona biológica. Genera una salida de 1 o 0, dependiendo de si su entrada efectiva excede un cierto valor, que se denomina valor **umbral** de la neurona. Esta entrada efectiva es una suma ponderada de las entradas reales, tal como se representa en la Figura 11.16. En esta figura, la neurona se presenta mediante un óvalo y las conexiones entre neuronas se representan mediante flechas. Los valores obtenidos de los axones de otras neuronas (designados mediante v_1, v_2, y v_3) se utilizan como entradas a la neurona mostrada. Además de estos valores, cada conexión tiene asociado un **peso** (designado mediante w_1, w_2 y w_3). La neurona que recibe estos valores de entrada multiplica cada uno de ellos por el peso asociado con esa conexión y luego suma dichos productos para obtener la entrada efectiva ($v_1w_1 + v_2w_2 + v_3w_3$). Si esta suma excede del valor umbral de la neurona, esta genera una salida igual a 1 (simulando un estado excitado); en caso contrario, la neurona genera un 0 como salida (simulando un estado inhibido).

Figura 11.16 Las actividades en el interior de una neurona.



De acuerdo con lo que se muestra en la Figura 11.16, adoptaremos el convenio de representar las neuronas como círculos. Allí donde se conecte cada entrada con una neurona anotaremos el peso asociado con dicha entrada. Finalmente, escribiremos el valor umbral de la neurona dentro del propio círculo. Por ejemplo, la Figura 11.17 representa una neurona con un valor umbral de 1,5 y con pesos de −2, 3 y −1 asociados con cada una de sus conexiones de entrada. Por tanto, si la neurona recibe las entradas 1, 1 y 0, su entrada efectiva será (1)(−2) + (1)(3) + (0)(−1) = 1, y su salida será igual a 0. Pero si la neurona recibe las entradas 0, 1 y 1, su entrada efectiva será (0)(−2) + (1)(3) + (1)(−1) = 2, que excede del valor umbral, por lo que la salida de la neurona pasará a ser 1.

El hecho de que un peso pueda ser positivo o negativo significa que la entrada correspondiente puede tener un efecto inhibidor o excitador sobre la neurona receptora. (Si el peso es negativo, entonces un 1 en esa posición de entrada reducirá la suma ponderada y tenderá por tanto a mantener la entrada efectiva por debajo del valor umbral. Por el contrario, un peso positivo hace que la entrada asociada tenga el efecto de incrementar la suma ponderada y aumentar, por tanto, la posibilidad de que dicha suma exceda del valor umbral.) Además, la magnitud real del peso controla el grado con el que la entrada correspondiente puede inhibir o excitar a la neurona receptora. En consecuencia, ajustando los valores de los pesos a través de una red neuronal artificial, podemos programar la red para que responda a diferentes entradas de una manera predeterminada.

Las redes neuronales artificiales suelen disponerse en una topología compuesta por varias capas. Las neuronas de entrada se encuentran en la primera capa y la neuronas de salida en la última. Pueden incluirse capas adicionales de neuronas (denominadas capas ocultas) entre las capas de entrada y de salida. Cada neurona de una capa está interconectada con todas las neuronas de la siguiente capa. Por ejemplo, la red simple mostrada en la Figura 11.18a está programada para generar una salida de 1 si sus dos entradas difieren y una salida igual a 0 en caso contrario. Sin embargo, si cambiamos los pesos y los sustituimos por los que se indican en la Figura 11.18b, obtenemos una red que responde con un 1 si ambas entradas son iguales a 1 y con un 0 en caso contrario.

Es preciso observar que la configuración de red de la Figura 11.18 es mucho más simple que la de una red biológica real. El cerebro humano contiene aproximadamente 10^{11} neuronas con unas 10^{4} sinapsis por neurona. De hecho, las dendritas de una neurona biológica son tan numerosas que se parecen más a una malla fibrosa que a los tentáculos individuales representados en la Figura 11.15.

Figura 11.17 Representación de una neurona.

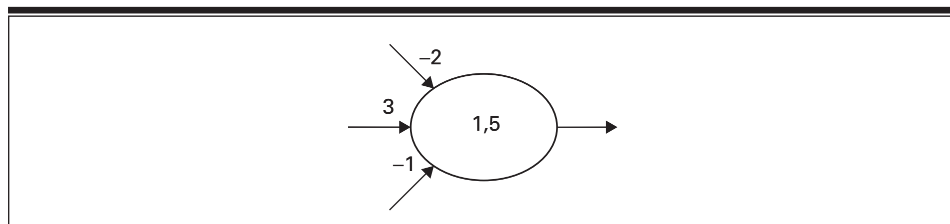
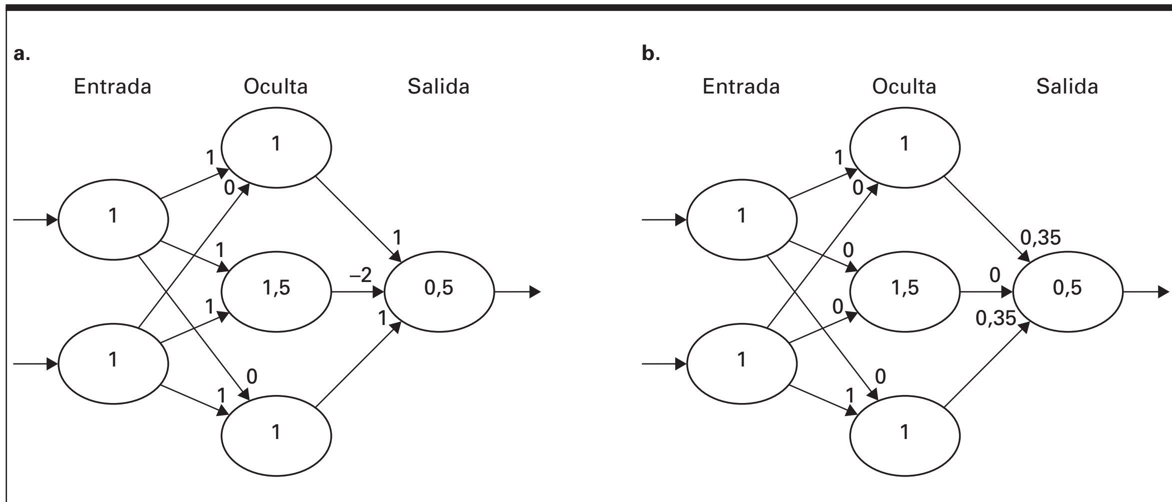


Figura 11.18 Una red neuronal con dos programas diferentes.



Entrenamiento con las redes neuronales artificiales

Una característica importante de las redes neuronales artificiales es que no se programan en el sentido tradicional, sino que en lugar de ello se las entrena. Es decir, el programador no determina los valores de los pesos necesarios para resolver un problema concreto y luego "inserta" esos valores en la red. En lugar de ello, una red neuronal artificial aprende cuáles son los valores apropiados de los pesos mediante un entrenamiento supervisado (Sección 11.4) que implica un proceso repetitivo en el que se aplican las entradas del conjunto de prueba a la red y luego los pesos se ajustan en incrementos pequeños, para que el comportamiento de la red se vaya aproximando al comportamiento deseado.

Es interesante observar que las técnicas de algoritmos genéricos se han aplicado a la tarea de entrenar redes neuronales artificiales. En particular, para entrenar una red neuronal, se puede generar un serie aleatoria de conjuntos de pesos para la red (cada conjunto actuará como un cromosoma del algoritmo genético). Después, en un proceso paso a paso, pueden asignarse a la red los pesos representados por cada cromosoma y probar la red con diversas entradas. Entonces puede asignarse a los cromosomas que generen el menor número de errores durante este proceso de prueba una mayor probabilidad de ser seleccionados como padres para la siguiente generación. En numerosos experimentos, esta técnica ha terminado por conducir a la obtención de un conjunto de pesos adecuado.

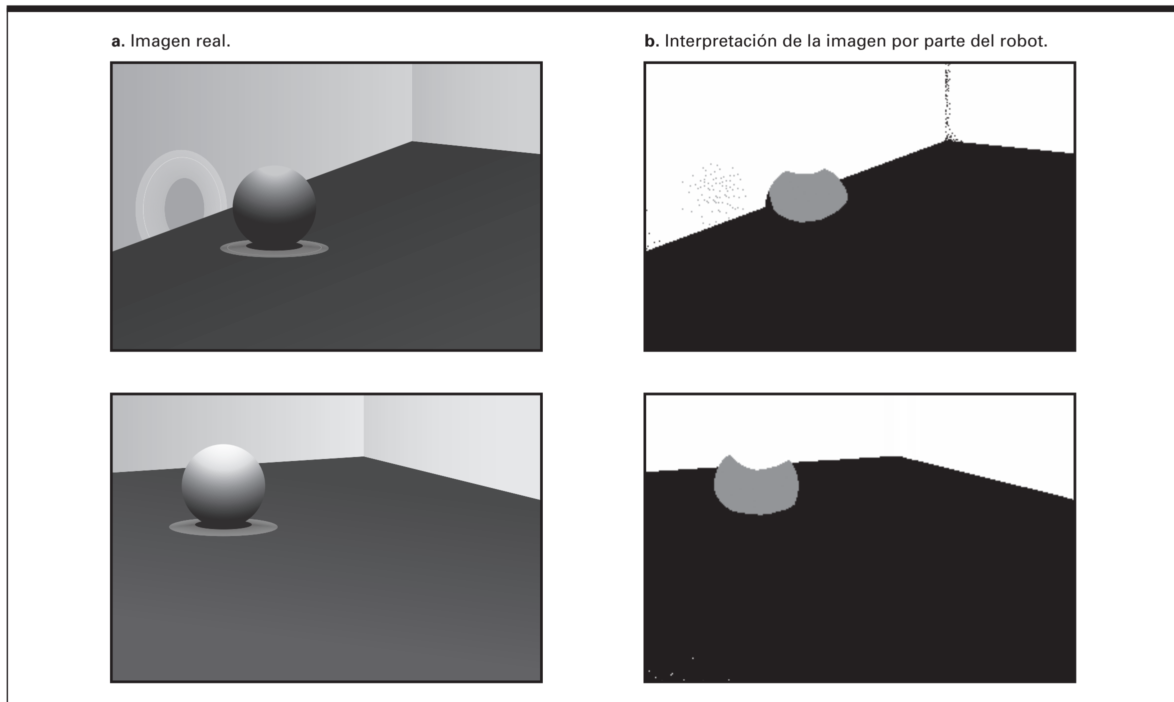
Consideremos un ejemplo en el que entrenar a una red neuronal artificial para resolver un problema ha tenido éxito y ha sido, quizá, más productivo que tratar de proporcionar una solución por medio de técnicas de programación tradicional. El problema es uno con el que un robot podría encontrarse al tratar de comprender su entorno gracias a la información recibida a través de una videocámara. Por ejemplo, suponga que el robot debe distinguir entre las paredes de una habitación, que son de color blanco, y el suelo, que es negro. A primera vista, parece que esto debería ser una tarea sencilla: bastaría con clasificar los píxeles blancos como parte de una pared y los pixeles negros como parte del

suelo. Sin embargo, a medida que el robot mira en diferentes direcciones o se mueve por la habitación, las distintas condiciones de iluminación pueden hacer que la pared parezca gris en algunos casos, mientras que en otras ocasiones es el suelo lo que puede parecer gris. Por tanto, el robot necesita aprender a distinguir las paredes y el suelo en una amplia variedad de condiciones de iluminación.

Para resolver esto, podríamos construir una red neuronal artificial cuyas entradas estuvieran compuestas por valores que indiquen las características de color de cada píxel individual de la imagen, así como por un valor que indique el brillo global de la imagen completa. Después, podríamos entrenar a la red proporcionándole numerosos ejemplos de píxeles que representen partes de las paredes y del suelo bajo distintas condiciones de iluminación.

Los resultados de entrenar a una red neuronal artificial con estas técnicas se ilustran en la Figura 11.19. La primera columna representa las imágenes originales, la siguiente muestra la interpretación realizada por el robot. Observe que, aunque las paredes en la imagen original de la parte superior son más bien oscuras, el robot ha identificado correctamente la mayoría de los píxeles asociados como píxeles blancos de una pared, sin por ello dejar de identificar correctamente el suelo presente en la imagen inferior (la bola presente en las imágenes era parte de un experimento más amplio). Observará también que el sistema de procesamiento de imágenes del robot no es perfecto. La red neuronal ha identificado incorrectamente algunos píxeles de la pared como píxeles del suelo (y algunos de los píxeles del suelo como píxeles de una pared). Estos son ejemplos de realidades que hay que tener en cuenta durante la aplicación

Figura 11.19 Resultados de utilizar una red neuronal para clasificar los píxeles de una imagen (inspirado en www.actapress.com)

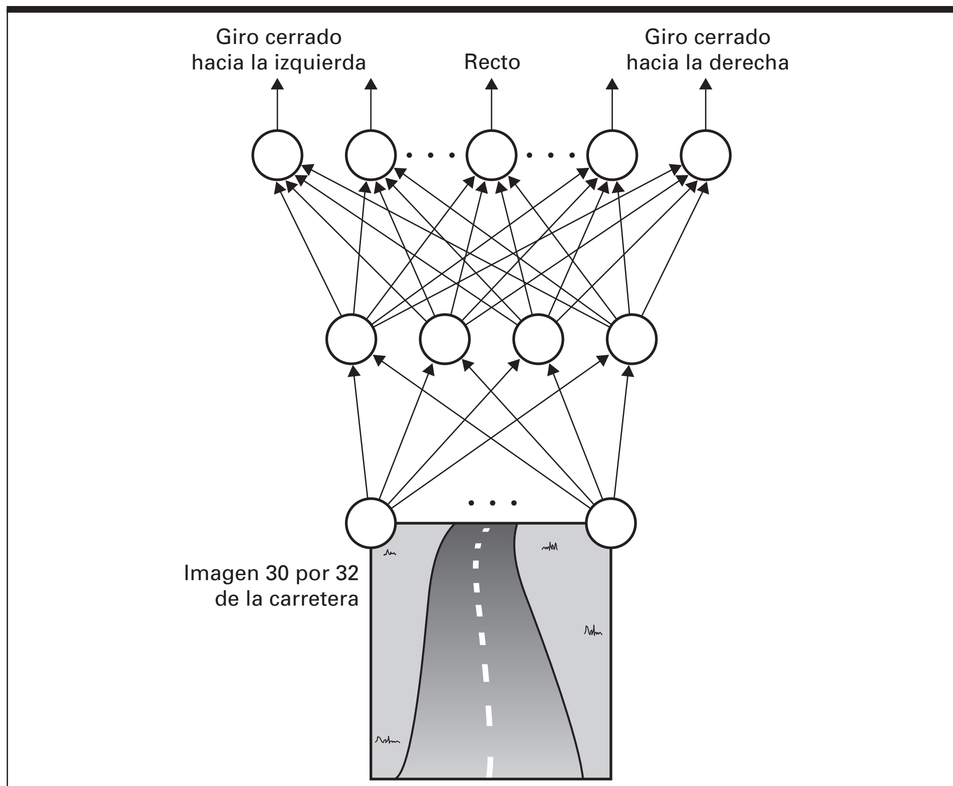


de una teoría. En este caso, los errores pueden corregirse programando al robot para que ignore los píxeles de suelo individuales que parezcan rodeados por una multitud de píxeles de pared (y viceversa).

Además de para problemas simples de aprendizaje (como el de clasificación de los píxeles), las redes neuronales artificiales se han estado utilizando para aprender comportamientos inteligentes sofisticados como atestigüa el proyecto ALVINN que hemos citado en la sección anterior. De hecho, ALVINN era una red neuronal artificial, cuya composición era tremendamente simple (Figura 11.20). Su entrada se obtenía a partir de una matriz de 30 por 32 sensores, cada uno de los cuales observaba una parte diferente de la imagen de vídeo de la carretera situada delante del vehículo e informaba de sus descubrimientos a cuatro neuronas situadas en una capa oculta (por tanto, cada una de estas cuatro neuronas tenía 960 entradas). La salida de cada una de estas cuatro neuronas estaba conectada a cada una de las 30 neuronas de salida, cuyas salidas indicaban la dirección en la que girar. La presencia de neuronas excitadas en uno de los extremos de la fila de 30 neuronas indicaba un giro cerrado hacia la izquierda, mientras que la presencia de neuronas excitadas en el otro extremo indicaba un giro cerrado hacia la derecha.

ALVINN era entrenado "observando" a un ser humano conducir, mientras que tomaba sus propias decisiones sobre hacia dónde girar, comparaba esas decisiones con las de la persona y realizaba pequeñas modificaciones en los pesos para hacer que sus decisiones se parecieran más a las que la persona

Figura 11.20 La estructura de ALVINN (Autonomous Land Vehicle in a Neural Net).



tomaba. Sin embargo, existe una cuestión colateral interesante. Aunque ALVINN aprendió a conducir mediante esta técnica simple, lo que no aprendió fue a recuperarse de los errores. Por tanto, los datos recopilados a partir de la observación del conductor humano se enriquecieron artificialmente con el fin de incluir también las situaciones en que hubiera que recuperarse de un error. Una solución para este entrenamiento de cara a la recuperación de errores que se adoptó inicialmente fue que el conductor humano efectuara virajes bruscos con el vehículo, para que ALVINN pudiera ver cómo recuperaba luego el control y aprendiera así a recuperar el control por su cuenta. Pero a menos que se desactivara a ALVINN mientras que el ser humano realizaba el giro brusco inicial, lo que ALVINN hacía era aprender también a realizar giros bruscos, además de a recuperar luego el control, lo que obviamente era una consecuencia indeseable.

Memoria asociativa

La mente humana tiene la increíble habilidad de poder extraer información que esté asociada con el tema que actualmente estemos considerando. Cuando percibimos ciertos olores, podemos fácilmente recordar cosas de nuestra infancia. El sonido de la voz de un amigo nos puede traer fácilmente a la mente la imagen de esa persona o quizá el recuerdo de algunos momentos agradables pasados en su compañía. Cierta música puede generar pensamientos relacionados con unas vacaciones concretas. Todos estos son ejemplos de **memoria asociativa**, la extracción de información que está asociada o relacionada con la información que en ese momento tenemos entre manos.

Construir máquinas con memoria asociativa ha sido un objetivo de investigación durante muchos años. Una aproximación consiste en aplicar técnicas de redes neuronales artificiales. Por ejemplo, considere una red compuesta por muchas neuronas que estén interconectadas para formar una especie de tela de araña sin entradas ni salidas. (En algunos diseños denominados redes de Hopfield, la salida de cada neurona se conecta como entrada a cada una de las otras neuronas; en otros casos, la salida de una neurona puede estar solo conectada a sus vecinas inmediatas.) En ese tipo de sistema, las neuronas excitadas tenderán a excitar a otras neuronas, mientras que las neuronas inhibidas tenderán a inhibir a otras. A su vez, el sistema completo puede encontrarse en un estado constante de cambio, o puede que termine por encontrar su camino hacia una configuración estable, en la que las neuronas excitadas permanecerán excitadas y las neuronas inhibidas permanecerán inhibidas. Si iniciamos la red en una configuración no estable que esté próxima a otra estable, cabría esperar que la red fuera derivando hacia esa configuración estable. En cierto sentido, cuando se le proporciona una parte de una configuración estable, la red puede ser capaz de completar esa configuración.

Ahora suponga que representamos un estado excitado mediante un 1 y un estado inhibido mediante un 0, de modo que la condición de toda la red en cualquier momento pueda representarse mediante una configuración de 0s y 1s. Entonces, si configuramos la red con un patrón de bits que esté próximo a un patrón estable cabría esperar que la red derivara hacia ese patrón estable. En otras palabras, la red podría encontrar el patrón estable de bits que esté próximo a ese patrón que se le ha proporcionado. De este modo, si algunos de los

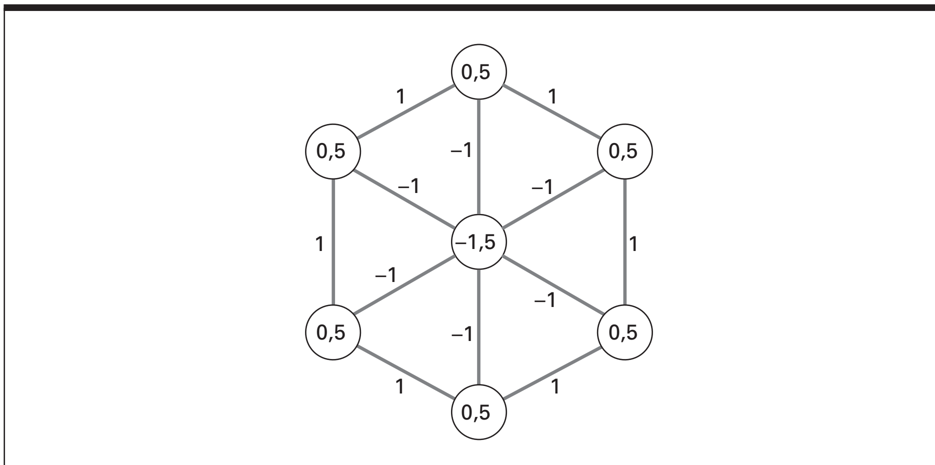
bits se emplean para codificar olores y otros se usan para codificar recuerdos de la infancia, entonces la inicialización de los bits correspondientes a los olores de acuerdo con una cierta configuración estable podría provocar que los bits restantes encontraran su camino hacia los recuerdos de infancia asociados.

Considere ahora la red neuronal artificial mostrada en la Figura 11.21. Siguiendo el convenio utilizado para dibujar redes neuronales artificiales, cada uno de los círculos de la figura representa una neurona cuyo valor umbral está anotado dentro del círculo. En lugar de flechas, las líneas que conectan los círculos representan conexiones bidireccionales entre las correspondientes neuronas. Es decir, si una línea conecta dos neuronas, eso indica que la salida de cada neurona está conectada como entrada de la otra. Por tanto, la salida de la neurona central está conectada como entrada a cada una de las neuronas situadas alrededor del perímetro, y la salida de cada una de las neuronas del perímetro está conectada como entrada a la neurona central además de estar conectada como entrada también a cada una de sus vecinas inmediatas del perímetro. Dos neuronas conectadas asociarán el mismo peso con la salida de la otra neurona. Este peso común se anota junto a la línea que conecta a ambas neuronas. Por tanto, la neurona situada en la parte superior del diagrama asocia un peso de −1 con la entrada que recibe de la neurona central y un peso de 1 con las entradas que recibe de sus dos vecinas del perímetro. De la misma forma, la neurona central asocia un peso de −1 con cada uno de los valores que recibe de cada una de las neuronas situadas alrededor del perímetro.

La red opera en pasos discretos en los que todas las neuronas responden a sus entradas de manera sincronizada. Para determinar la siguiente configuración de la red a partir de su configuración actual, calculamos las entradas efectivas de cada neurona de la red y luego permitimos que todas las neuronas respondan a sus entradas simultáneamente. El efecto es que toda la red sigue una secuencia coordinada en la que se calculan las entradas efectivas, se responde a las entradas, se calculan de nuevo la entradas efectivas, se responde otra vez a las entradas, etc.

Considere la secuencia de sucesos que tendría lugar si inicializáramos la red con las dos neuronas de la derecha inhibidas y las otras neuronas excitadas

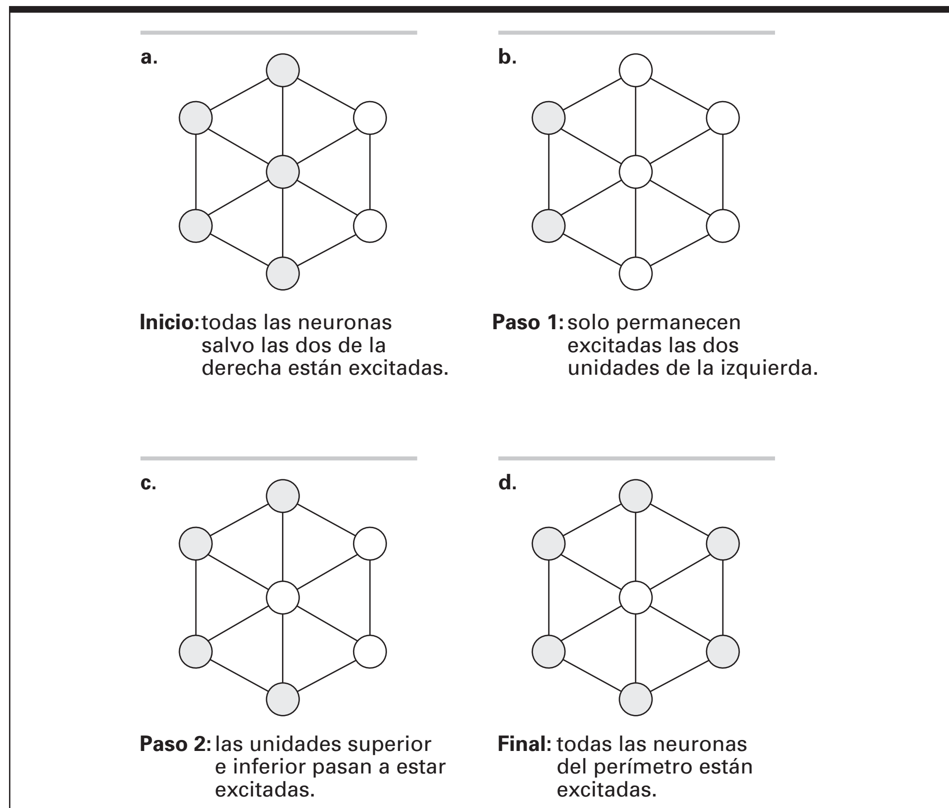
Figura 11.21 Una red neuronal artificial que implementa una memoria asociativa.



(Figura 11.22a). Las dos neuronas de la izquierda tendrán una entrada efectiva igual a 1, así que continuarán excitadas. Pero sus vecinas del perímetro tendrán una entrada efectiva igual a 0, por lo que quedarán inhibidas. De la misma forma, la neurona central tendrá una entrada efectiva de −4, así que también se inhibirá. Por tanto, toda la red se desplazará a la configuración mostrada en la Figura 11.22b, en la que solo aparecen excitadas las dos neuronas de la izquierda. Puesto que la neurona central estaría ahora inhibida, la condición de excitadas de las neuronas de la izquierda hará que las neuronas superior e inferior vuelvan a excitarse. Mientras tanto, la neurona central seguirá inhibida, ya que tendrá una entrada efectiva igual a −2. Por tanto, la red pasará a la configuración mostrada en la Figura 11.22c, de la cual después pasará a la configuración ilustrada en la Figura 11.22d. Pruebe a confirmar que si la red se inicializa con solo las cuatro neuronas superiores excitadas, se producirá un fenómeno de parpadeo. La neurona superior permanecerá excitada, mientras que su dos vecinas del perímetro y la neurona central irán alternando entre el estado excitado y el inhibido.

Por último, observe que la red tiene dos configuraciones estables: una en la que la neurona central está excitada y las demás están inhibidas y otra en la que la neurona central está inhibida y las restantes neuronas están excitadas. Si inicializamos la red con la neurona central excitada y no más de dos de las otras neuronas excitadas, la red irá derivando hacia la primera de esas dos con-

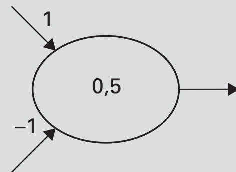
Figura 11.22 Los pasos que conducen a una configuración estable.



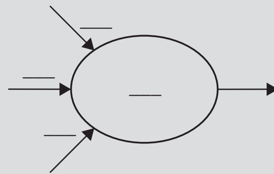
figuraciones estables. Si inicializamos la red con al menos cuatro neuronas adyacentes del perímetro en su estado excitado, la red derivará hacia la segunda configuración estable. Por tanto, podemos decir que la red asocia la primera de las configuraciones estables con aquellos patrones iniciales en los que su neurona central y menos de tres de sus neuronas perimetrales están excitadas, mientras que asocia la segunda configuración estable con aquellos patrones iniciales en los que hay cuatro o más de las neuronas perimetrales excitadas. En resumen, la red representa una memoria asociativa de carácter elemental.

Cuestiones y ejercicios

1. ¿Cuál será la salida de la siguiente neurona cuando sus dos entradas sean iguales a 1? ¿Y qué pasa con los patrones de entrada 0, 0; 0, 1; y 1, 0?



2. Ajuste los pesos y el valor umbral de la siguiente neurona para que su salida sea 1 si y solo si al menos dos de sus entradas son iguales a 1.



3. Identifique un problema que puede surgir a la hora de entrenar una red neuronal artificial.
4. ¿Hacia qué configuración estable derivará la red de la Figura 11.22 si se inicializa con todas sus neuronas inhibidas?

11.6 Robótica

La **robótica** es el estudio de los agentes físicos autónomos que se comportan de manera inteligente. Al igual que sucede con cualquier otro agente, los robots deben ser capaces de percibir, razonar y actuar dentro de su entorno. Las investigaciones en robótica abarcan, por tanto, todas las áreas de la inteligencia artificial, además de utilizar intensamente los resultados de la ingeniería mecánica y eléctrica.

Para interactuar con el mundo, los robots necesitan mecanismos para manipular los objetos y para desplazarse. En los primeros días de la robótica, el campo estaba estrechamente relacionado con el desarrollo de manipuladores, que muy a menudo eran brazos mecánicos con codos, muñecas y

manos o herramientas. La investigación no trataba solo de determinar cómo podían maniobrarse esos dispositivos, sino que también intentaba ver cómo podía mantenerse y aplicarse el conocimiento acerca de su posición y de su orientación. (Los seres humanos somos capaces de cerrar los ojos y seguir tocándonos la nariz con el dedo, porque nuestro cerebro mantiene un registro de dónde se encuentran nuestros dedos y nuestra nariz.) Con el tiempo, los brazos de robot se han hecho mucho más sofisticados, hasta el punto de que, con un cierto sentido del tacto basado en la realimentación dinámica, pueden manejar huevos o tazas de papel sin ningún problema.

Recientemente, el desarrollo de computadoras más rápidas y ligeras ha dado un impulso a las investigaciones en el campo de los robots móviles que pueden desplazarse. Conseguir esta movilidad ha conducido a una gran abundancia de diseños creativos. Los investigadores en el campo de la locomoción de los robots han desarrollado robots que nadan como los peces, que vuelan como las libélulas, que saltan como los saltamontes y que reptan como las serpientes.

Los robots con ruedas son muy populares ya que son relativamente fáciles de diseñar y de construir, pero están limitados en cuanto al tipo de terreno por el que se pueden desplazar. Sobreponerse a estas restricciones, utilizando combinaciones de ruedas o cadenas para subir escaleras o para moverse por terrenos rocosos es el objetivo de muchas de las investigaciones actuales. Por ejemplo, los exploradores marcianos de la NASA utilizan ruedas especialmente diseñadas para moverse por suelo rocoso.

Los robots con piernas ofrecen una mayor movilidad, pero son bastante más complejos. Por ejemplo, los robots con dos patas diseñados para caminar como los seres humanos, deben constantemente monitorizar y ajustar su posición para mantener el equilibrio, ya que de lo contrario se caen. Sin embargo, dichas dificultades pueden ser resueltas como ilustra el robot humanoide de dos piernas, denominado Asimo, desarrollado por Honda, que puede subir escaleras e incluso correr.

A pesar de los grandes avances en el campo de los manipuladores y de la locomoción, la mayoría de los robots todavía no son muy autónomos. Los bra-

Robots que han hecho historia

a. El robot "Boss" de Tartan Racing, ganador del Urban Challenge (desafío urbano), un concurso patrocinado por DARPA, en el que los vehículos tienen que conducir por sí mismos en un entorno urbano (© DARPA). **b.** Uno de los exploradores de la NASA, un robot geólogo que explora la superficie de Marte (Cortesía de NASA/JPL-Caltech).

a.



b.



zos robóticos industriales están, normalmente, rígidamente programados para cada tarea y trabajan sin sensores, suponiendo que los componentes se les entregarán en posiciones exactas. Otros robots móviles, como los exploradores marcianos de la NASA y los vehículos aéreos no tripulados de carácter militar dependen de los operadores humanos en lo que a inteligencia se refiere.

Sobreponerse a esta dependencia de los seres humanos es uno de los principales objetivos de las investigaciones actuales. Una de las cuestiones que se plantean trata con lo que un robot autónomo necesita conocer acerca de su entorno y con el grado de antelación con el que necesita planificar sus acciones. Un enfoque consiste en construir robots que mantengan registros detallados de sus entornos, que contengan un inventario de objetos junto con sus posiciones con el que desarrollar planes de acción precisos. Las investigaciones en esta dirección dependen en gran medida del progreso que se realice en el campo de la representación y almacenamiento del conocimiento, así como del desarrollo de técnicas mejoradas de razonamiento y de desarrollo de planes.

Un enfoque alternativo consiste en desarrollar robots reactivos que, en lugar de mantener registros complejos e invertir un gran esfuerzo en construir planes de acción detallados, simplemente aplican reglas sencillas de interacción con el mundo para guiar su comportamiento en cada momento. Los defensores de la robótica reactiva argumentan que a la hora de planificar un viaje largo en un vehículo, los seres humanos no hacen planes detallados de antemano en los que se contemplen todas las alternativas. En lugar de ello, simplemente seleccionan las carreteras principales, dejando para luego detalles tales como dónde comer, qué salidas tomar y cómo reaccionar en el caso de encontrarse con un desvío. De la misma forma, un robot reactivo que necesite desplazarse por un pasillo atestado o que necesite ir de un edificio a otro no tiene por qué desarrollar de antemano un plan enormemente detallado, sino que simplemente tiene que aplicar una serie de reglas sencillas para evitar cada obstáculo a medida que se lo encuentra. Este es el enfoque adoptado por el robot más vendido de la historia, el robot aspiradora iRobot Roomba, que se mueve por el suelo en modo reactivo sin preocuparse de recordar los detalles acerca de los muebles y otros obstáculos. Después de todo, lo más probable es que la mascota de la familia no se encuentre en el mismo lugar la siguiente vez.

Por supuesto, no es probable que ninguna solución resulte ser la óptima para todas las situaciones posibles. Los robots verdaderamente autónomos utilizarán, probablemente, múltiples niveles de razonamiento y de planificación, aplicando técnicas de alto nivel para establecer y conseguir los objetivos principales y sistemas reactivos de bajo nivel para lograr los subobjetivos secundarios. Un ejemplo de este tipo de razonamiento multinivel lo podemos encontrar en el concurso Robocup, un concurso internacional de equipos de fútbol robóticos, que sirve como foro para las investigaciones tendentes a desarrollar equipos de robots que puedan jugar mejor que los equipos profesionales formados por seres humanos en el año 2050. Aquí, el énfasis no está solo en construir robots móviles que puedan "patear" un balón, sino en diseñar un equipo de robots que puedan cooperar unos con otros con el fin de conseguir un objetivo común. Estos robots no solo tienen que desplazarse y razonar acerca de sus acciones, sino que también tienen que razonar acerca de las acciones de sus compañeros y de sus oponentes.

Otro ejemplo de investigación en el campo de la robótica es el subcampo conocido como robótica evolutiva, en el que se aplican las teorías de la evolución para el desarrollo de esquemas tanto de reglas reactivas de bajo nivel como razonamiento de alto nivel. En este subcampo nos encontramos con que la teoría de la supervivencia del más adaptado se utiliza para desarrollar dispositivos que a lo largo de múltiples generaciones adquieren sus propios medios de equilibrio o de movilidad. Buena parte de la investigación en esta área distingue entre el sistema de control interno de un robot (que es fundamentalmente software) y la estructura física de su cuerpo. Por ejemplo, el sistema de control de un robot renacuajo nadador fue transferido a un robot similar que tenía piernas. Entonces, se aplicaron técnicas evolutivas en el sistema de control para obtener un robot capaz de reptar. En otros casos se han aplicado técnicas evolutivas al cuerpo físico de un robot para descubrir las posiciones óptimas de los sensores para la realización de una tarea concreta. Otras investigaciones más complejas tratan de encontrar formas para hacer evolucionar los sistemas de control software de manera simultánea con las estructuras físicas del cuerpo.

Enumerar todo el conjunto de impresionantes resultados generados por las investigaciones en el campo de la robótica sería una tarea abrumadora. Los robots actuales están todavía lejos de esos potentes robots que se presentan en las novelas y películas de ficción, pero se han conseguido éxitos impresionantes en tareas específicas. Disponemos de robots que pueden conducir en condiciones de tráfico intenso, comportarse como mascotas y guiar misiles hasta su objetivo. Sin embargo, aunque nos alegremos de estos éxitos, es preciso observar que el cariño que sentimos por un perro artificial y el increíble poder de las armas inteligentes plantean cuestiones sociales y éticas que representan un auténtico desafío para nuestra sociedad. Nuestro futuro será tal como nosotros lo construyamos.

Cuestiones y ejercicios

1. ¿En qué sentido difiere la solución reactiva para fijar el comportamiento de un robot del comportamiento más tradicional "basado en planes"?
2. ¿Podría citar algunos de los temas actuales de investigación en el campo de la robótica?
3. ¿Podría citar dos niveles en los que se estén aplicando teorías evolutivas para el desarrollo de robots?

11.7 Consideración de las consecuencias

Sin ninguna duda, los avances que se están realizando en el campo de la inteligencia artificial ofrecen el potencial de beneficiar a la humanidad, y es fácil verse atrapado por el entusiasmo que generan esos potenciales beneficios. Sin embargo, también hay peligros potenciales acechando en el futuro, cuyas ramificaciones podrían ser tan devastadoras como beneficiosas son sus contrapartidas. La distinción está a menudo, simplemente, en el punto de vista de cada uno o quizá en la posición de cada uno dentro de la sociedad, las ventajas obte-

nidas por una persona pueden ser las pérdidas de otra. Es conveniente, por tanto, que dediquemos unos instantes a examinar estos avances de la tecnología desde algunas perspectivas alternativas.

Algunas personas ven los avances de la tecnología como un regalo para la humanidad, un medio de librar a las personas de las aburridas tareas mundanas y de abrir la puerta a estilos de vida más placenteros. Pero otras personas ven este mismo fenómeno como una maldición que quita empleos a los ciudadanos y canaliza la riqueza hacia aquellos que disponen del poder. De hecho, este era uno de los mensajes de Mahatma Gandhi en la India. Él insistía una y otra vez que para la India sería mejor sustituir las grandes fábricas de tejidos por ruecas instaladas en las casas de los trabajadores. De esta forma, decía, la producción centralizada en masa que solo da empleo a unos cuantos se vería sustituido por un sistema de distribución masivo que beneficiaría a las multitudes.

La historia está llena de revoluciones que hunden sus raíces en la desproporcionada distribución de la riqueza y los privilegios. Si se permite que los actuales avances tecnológicos aumenten esas diferencias, las consecuencias podrían ser catastróficas.

Pero las consecuencias de construir máquinas cada vez más inteligentes son más sutiles, más fundamentales, que las que se refieren a las luchas de poder entre distintos segmentos de la sociedad. Los problemas que se plantean afectan al auténtico núcleo de la imagen que la humanidad tiene de sí misma. En el siglo XIX, la sociedad se escandalizó por la teoría de la evolución de Charles Darwin y por el pensamiento de que los seres humanos podían haber evolucionado a partir de formas de vida menos complejas. ¿Cómo reaccionará entonces la sociedad si se la enfrenta con el fenómeno de la aparición de máquinas cuyas capacidades mentales desafíen las de los propios seres humanos?

En el pasado, la tecnología se ha desarrollado lentamente, dejando así tiempo para que la imagen que tenemos de nosotros mismos quedara preservada reajustando nuestro concepto de la inteligencia. Nuestros antepasados de hace muchos siglos habrían interpretado los dispositivos mecánicos del siglo XIX como dotados de una inteligencia sobrenatural, pero hoy día no concedemos a esas máquinas ninguna inteligencia en absoluto. ¿Pero cómo reaccionará la humanidad si las máquinas comienzan verdaderamente a desafiar la inteligencia de los humanos o, más probablemente, si las capacidades de las máquinas comienzan a avanzar más rápidamente que nuestra capacidad de adaptación?

Una pista acerca de cuál sería la reacción potencial de la humanidad a la aparición de máquinas que desafíen nuestro intelecto nos la ofrece el análisis de cuál fue la respuesta social a los test de inteligencia a mediados del siglo XX. Esos test se consideraron para tratar de identificar el nivel de inteligencia de los niños. Los niños en Estados Unidos se clasificaron según su rendimiento en este tipo de test y se les dirigió hacia los distintos programas educativos de acuerdo con los resultados. Esto hizo que se abrieran oportunidades educativas para aquellos niños que obtenían buenos resultados en esos test, mientras que los niños con resultados mediocres se canalizaban hacia programas de estudio compensatorios. En resumen, al proporcionarle una escala con la que medir la inteligencia, la sociedad tendió a no considerar las capacidades de aquellos que se vieron situados en el extremo inferior de la escala. ¿Cómo gestionaría entonces nuestra sociedad la situación si las capacidades "intelectuales" de las máquinas pasaran a ser comparables o incluso parecieran comparables, a las de los huma-

nos? ¿Volvería la espalda nuestra sociedad a aquellos cuyas capacidades fueran vistas como "inferiores" a las de las máquinas? En caso afirmativo, ¿cuáles serían las consecuencias para esos miembros de la sociedad? ¿Debería estar sujeta la dignidad de una persona a cómo se compare esa persona con una máquina?

Ya hemos comenzado a ver cómo la capacidad intelectual de los seres humanos se ve desafiada por las máquinas en una serie de campos específicos. Las máquinas son capaces ahora de derrotar a los jugadores expertos de ajedrez; existen sistemas expertos computerizados que son capaces de proporcionar consejos de tipo médico y existen programas simples de gestión de carteras de inversión que a menudo obtienen resultados mejores que los de los inversores profesionales. ¿Cómo afectan esos sistemas a la imagen que tienen de sí mismos los individuos afectados? ¿Cómo se verá afectada la autoestima de una persona al verse desplazada por las máquinas en un número cada vez mayor de áreas?

Muchos argumentan que la inteligencia que poseen las máquinas siempre será inherentemente distinta de la de los seres humanos, ya que los humanos somos entes biológicos y las máquinas no. Por tanto, argumentan que las máquinas no serán nunca capaces de reproducir el proceso de toma de decisiones de los seres humanos. Las máquinas podrían tomar las mismas decisiones que los humanos, pero esas decisiones no se realizarían basándose en las mismas cosas que los seres humanos. ¿Hasta qué grado existen entonces diferentes tipos de inteligencia y hasta qué grado sería ético que nuestra sociedad siguiera las directrices propuestas por una inteligencia no humana?

En su libro, ***Computer Power and Human Reason,*** Joseph Weizenbaum argumenta en contra de la aplicación indiscriminada de la inteligencia artificial diciendo lo siguiente:

> Las computadoras pueden tomar decisiones judiciales. Las computadoras pueden emitir opiniones de carácter psiquiátrico. Pueden lanzar monedas al aire en formas mucho más sofisticadas que los seres humanos más pacientes, pero lo importante es que no ***deberíamos*** asignarles dichas tareas. Puede incluso que sean capaces de llegar a tomar decisiones "correctas" en algunos casos, pero siempre y necesariamente sobre bases que ningún ser humano debería estar dispuesto a aceptar.
>
> Ha habido numerosos debates acerca de las "computadoras y la mente". Como conclusión me gustaría decir aquí que los problemas relevantes no son tecnológicos ni tampoco matemáticos; se trata de problemas éticos, problemas que no pueden solucionarse planteando preguntas que comiencen con "podemos". Los límites de la aplicabilidad de las computadoras solo pueden enunciarse, en definitiva, en términos de deberes. La conclusión más elemental de todo esto es que, puesto que ahora no disponemos de ninguna forma de hacer sabias a las computadoras, no deberíamos asignarles tareas que requieran sabiduría.

El lector podría argumentar que buena parte de esta sección bordea la ciencia ficción más que las Ciencias de la computación. Sin embargo, no hace tanto tiempo muchas personas descartaban la pregunta "¿Qué sucederá si las computadoras se hacen con el control de la sociedad?" con la misma aptitud de "eso no sucederá nunca". Pero en muchos aspectos, ese día ya ha llegado.

Si una base de datos computerizada informara erróneamente de que nuestro límite de crédito es muy bajo, de que poseemos antecedentes penales o de que nuestra cuenta bancaria no tiene saldo, ¿qué es lo que prevalecerá, la afirmación hecha por la computadora o nuestras demandas de inocencia? Si un sistema de navegación que funcione mal indica que una pista cubierta por la niebla se encuentra en el lugar incorrecto, ¿dónde aterrizará el avión? Si se emplea una máquina para predecir la reacción de la opinión pública a diversas decisiones políticas, ¿qué decisión tomará un político? ¿Cuántas veces se ha encontrado usted en esa situación en la que un administrativo le dice que no puede ayudarle porque "la computadora no funciona"? ¿Quién (o qué) está entonces a cargo de las cosas? ¿Acaso no hemos ya entregado nuestra sociedad a las máquinas?

Cuestiones y ejercicios

1. ¿Qué parte de la población actual sobreviviría si nos deshiciéramos de las máquinas desarrolladas a lo largo de los últimos cien años? ¿Y a lo largo de los últimos cincuenta años? ¿Y a lo largo de los últimos veinte años? ¿Dónde estarían localizados los supervivientes?
2. ¿Hasta qué grado está su vida controlada por las máquinas? ¿Quién controla las máquinas que afectan a su vida?
3. ¿Dónde obtiene la información en la que basa sus decisiones cotidianas? ¿Y sus decisiones más importantes? ¿Qué confianza tiene en la precisión de esa información? ¿Por qué?

Problemas de repaso

(Los problemas marcados con asterisco están asociados con las secciones opcionales.)

1. Como se ha ilustrado en la Sección 11.2, las personas pueden utilizar una pregunta con propósitos distintos del de realmente preguntar. Por ejemplo, la frase "¿Sabes que tienes una rueda pinchada?" se utilizaría más bien para informar a alguien que para preguntarle. Proporcione ejemplos de preguntas utilizadas para reafirmar, advertir y criticar.
2. Analice un cajero automático como si fuera un agente. ¿Cuáles son sus sensores? ¿Cuáles son sus actuadores? ¿Qué nivel de respuesta exhibe (refleja, basada en conocimientos, basada en objetivos)?
3. Identifique cada una de las siguientes respuestas como refleja, basada en conocimientos o basada en objetivos. Justifique sus respuestas.
   a. Un partido de tenis.
   b. Escalar una montaña.
   c. Una partida de ajedrez.
4. ¿Cómo se analiza el lenguaje natural?
5. ¿Cómo se clasifican los diferentes métodos de aprendizaje?

*6. ¿Cuál es la diferencia entre el conocimiento procedimental y el conocimiento declarativo? Explique su respuesta utilizando un ejemplo.

7. ¿Cuáles de las siguientes actividades esperaría que estuvieran orientadas al rendimiento y cuáles orientadas a la simulación?

a. El diseño de un sistema de vehículos de transporte automatizado (empleado a menudo en aeropuertos para trasladarse de una terminal a otra).

b. El diseño de un modelo que predice la ruta seguida por un huracán.

c. El diseño de una base de datos de búsqueda en la Web utilizada para crear y mantener índices de documentos almacenados en la World Wide Web

d. El diseño de un modelo de la economía de una nación, con el fin de probar teorías.

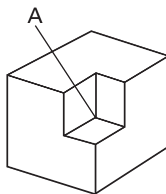
e. El diseño de un programa para monitorizar los signos vitales de un paciente.

8. ¿Puede aplicarse el test de Turing a un sistema en el que la entrada y la salida están restringidas? ¿Puede aplicarse si el sistema solo toma datos binarios y proporciona resultados en formato binario?

9. Identifique un pequeño conjunto de propiedades geométricas que pueda utilizarse para distinguir entre los símbolos F, E, L y T.

***10.** ¿Qué es una red neuronal artificial?

11. Describa dos interpretaciones del siguiente dibujo lineal, basándose en si la "esquina" marcada por la letra "A" es convexa o cóncava:



12. Compare los papeles de las cláusulas preposicionales incluidas en las dos frases siguientes (que sólo difieren en una palabra). ¿Cómo podría programarse una máquina para poder hacer tales distinciones?

La pared fue construida por el viento.
La pared fue construida por el granjero.

13. ¿Cómo difieren los resultados de analizar sintácticamente las dos frases siguientes? ¿Cómo difieren los resultados del análisis semántico?

Una maravillosa puesta de sol fue contemplada por Andrea.
Andrea contempló una maravillosa puesta de sol.

14. ¿Cómo difieren los resultados de analizar sintácticamente las dos frases siguientes? ¿Cómo difieren los resultados del análisis semántico?

Si X < 10 then restar 1 de X else sumar 1 a X.
Si X > 10 then sumar 1 a X else restar 1 de X.

15. En el texto hemos explicado brevemente los problemas de comprender los lenguajes naturales, por oposición a los lenguajes de programación formales. Como ejemplo de las complejidades en el caso de los lenguajes naturales, identifique situaciones en las que la pregunta "¿Sabes qué hora es?" tenga diferentes significados.

16. Los cambios en el contexto de una frase pueden variar la importancia de la frase además de su significado. En el contexto de la Figura 11.3, ¿cómo cambiaría la importancia de la frase "María golpeó a Juan" si las fechas de nacimiento estuvieran ambas a finales de la década de 2000? ¿Y si una estuviera en la década de 1980 y otra a finales de la década de 2000?

17. Dibuje una red semántica que represente la información contenida en el siguiente párrafo:

Diana arrojó la pelota a Joaquín, que la lanzó hacia el centro del campo. El centrocampista trató de atraparla, pero en lugar de ello rebotó en la pared.

18. En ocasiones, la capacidad de responder a una pregunta depende tanto de conocer los límites del conocimiento, como de los propios hechos. Por ejemplo, suponga que las bases de datos A y B contienen, las dos, una lista completa de empleados adheridos al

programa de seguro médico de la empresa, pero solo la base de datos A es consciente de que la lista es completa. ¿Qué podría concluir la base de datos A acerca de un empleado que no se encuentre en su lista, mientras que la base de datos B no podría llegar a esa conclusión?

19. ¿Cuáles son los componentes de un sistema de producción?

20. ¿Cuáles son las distintas formas en las que puede construirse un árbol de búsqueda?

21. ¿Qué es un agente inteligente?

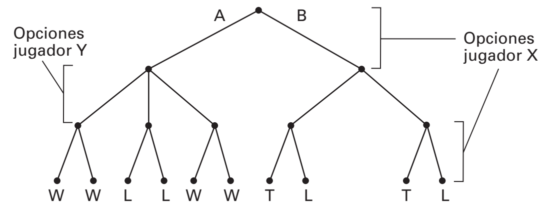
22. Analice la tarea de resolver un sudoku en términos de un sistema de producción. (¿Cuáles son los estados, las producciones, etc.?)

23. a. Suponga que un árbol de búsqueda es un árbol binario y que alcanzar el objetivo requiere ocho producciones. ¿Cuál es el máximo número de nodos que podría tener el árbol en el momento de alcanzar el estado objetivo, si se construye el árbol con una búsqueda en anchura?

b. Explique cómo se podría reducir el número total de nodos analizados durante la búsqueda realizando dos búsquedas al mismo tiempo: una que comience en el estado inicial y otra que busque hacia atrás desde el estado objetivo, hasta que las dos se encuentren. (Suponga que el árbol de búsqueda en el que se anotan los estados que vamos encontrando en la búsqueda hacia atrás es también un árbol binario y que ambas búsquedas progresan al mismo ritmo.)

24. En el texto hemos mencionado que los sistemas de producción se emplean a menudo como técnica para extraer conclusiones de hechos conocidos. Los estados del sistema son los hechos que sabemos que son ciertos en cada etapa del proceso de razonamiento y las producciones son las reglas de la lógica utilizadas para manipular los hechos conocidos. Identifique algunas reglas de la lógica que permiten extraer la conclusión "Juan es alto" a partir de los hechos "Juan es un jugador de baloncesto", "Los jugadores de baloncesto no son bajos" y "Juan es o bajo o alto".

25. El siguiente árbol representa los posibles movimientos en un juego de competición.



En este árbol podemos ver que el jugador X tiene actualmente la opción de efectuar el movimiento A o el movimiento B. Después del movimiento del jugador X, el jugador Y puede seleccionar un movimiento y entonces el jugador X puede elegir el último movimiento de la partida. Los nodos hoja del árbol están etiquetados como W, L o T, dependiendo de si ese final representa una victoria (W), una derrota (L) o una situación de tablas (T) para el jugador X. ¿Qué debería seleccionar el jugador X, el movimiento A o el B? ¿Por qué? ¿Cómo difiere la selección de una producción en un entorno competitivo de ese mismo proceso de seleccionar una producción en un juego para una sola persona como es el puzzle de ocho piezas?

26. Analice el juego de damas como un sistema de producción y describa un heurístico que podría utilizarse para determinar cuál de dos estados está más próximo al objetivo. ¿Cómo difiere el sistema de control de este juego del utilizado en un juego unipersonal, como por ejemplo el puzzle de ocho piezas?

27. Considerando la reglas de manipulación del álgebra como producciones, los problemas relativos a la simplificación de expresiones algebraicas pueden resolverse en el contexto de un sistema de producción. Identifique un conjunto de producciones algebraicas que permitan reducir la ecuación $3/(2x - 1) = 6/(3x + 1)$ a la forma $x = 3$. ¿Podría indicar algunas reglas prácticas (es decir, reglas heurísticas) utilizadas

al realizar ese tipo de simplificaciones algebraicas?

28. Dibuje el árbol de búsqueda generado por una búsqueda en anchura que trate de resolver el puzzle de ocho piezas a partir del siguiente estado inicial, sin ayuda de ninguna información heurística.

	1	3
4	2	5
7	8	6

29. Dibuje el árbol de búsqueda que generará el algoritmo de búsqueda por elección del mejor de la Figura 11.10 al intentar resolver el puzzle de ocho piezas a partir del estado inicial mostrado en el Problema 28, si empleamos como heurístico el número de piezas descolocadas

30. Dibuje el árbol de búsqueda que se genera mediante el algoritmo de búsqueda por elección del mejor de la Figura 11.10 al intentar resolver el puzzle de ocho piezas a partir del siguiente estado inicial, suponiendo que el heurístico utilizado es el mismo que el desarrollado en la Sección 11.3.

1	2	3
5	7	6
4		8

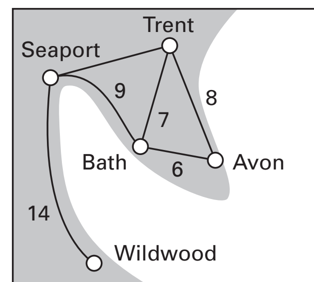
31. Al resolver el puzzle de ocho piezas, ¿por qué el número de piezas descolocadas no es tan buen heurístico como el que hemos empleado en la Sección 11.3?

32. ¿Cuál es la diferencia entre la técnica de decidir qué mitad de la lista hay que usar a la hora de realizar una búsqueda binaria (Sección 5.5) y la técnica de decidir qué rama explorar al llevar a cabo una búsqueda heurística?

33. Observe que si un estado del grafo de estados de un sistema de producción tiene un valor heurístico extremadamente bajo en comparación con los otros estados y si existe una producción que va desde ese estado a sí mismo, el algoritmo de la Figura 11.10 puede verse atrapado en un bucle en el que estaría tomando en consideración este estado una y otra vez. Demuestre que si el coste de ejecutar cualquier producción en el sistema es al menos uno, entonces al calcular el coste estimado como la suma del valor heurístico más el coste de alcanzar el estado en cuestión a lo largo de la ruta que se ha recorrido, se evitaría este proceso de bucle infinito.

34. ¿Qué heurístico utilizaría una computadora a la hora de jugar una partida a las tres en raya con una persona?

35. Dibuje hasta cuatro niveles del árbol de búsqueda generado por el algoritmo de búsqueda por elección del mejor de la Figura 11.10 a la hora de averiguar la ruta entre Trent y Wildwood en el siguiente mapa. Cada nodo del árbol de búsqueda será una ciudad del mapa. Comience con el nodo correspondiente a Trent. A la hora de expandir un nodo, añada solo las ciudades que estén directamente conectadas a la ciudad que se está expandiendo. Anote en cada nodo la distancia en línea recta hasta Wildwood y utilice ese valor como heurístico. ¿Tiene algún defecto el algoritmo de búsqueda por elección del mejor con este enfoque? En caso afirmativo, ¿qué corrección hace falta?



Distancia en línea recta a Wildwood desde

Avon	10
Bath	8
Trent	15
Seaport	13

36. El algoritmo A* modifica el algoritmo de búsqueda por elección del mejor de dos formas significativas. En primer lugar, anota el coste real necesario para alcanzar cada estado. En el caso de una ruta en un mapa, el coste real es la distancia recorrida. En segundo lugar, a la hora de seleccionar el nodo que hay que expandir, elige el nodo

que tenga un valor mínimo de la suma entre el coste real y el valor heurístico. Dibuje el árbol de búsqueda del Problema 35 que resultaría al efectuar estas dos modificaciones. Anote en cada nodo la distancia recorrida hasta la ciudad, el valor heurístico para alcanzar el objetivo y su suma. ¿Cuál es la ruta encontrada desde Dearborn a Wildwood?

37. ¿Cuáles son las diferencias entre la inteligencia artificial fuerte y la inteligencia artificial débil?

38. En un determinado juego para dos jugadores, se tiene una pila de 5 peniques colocada sobre el suelo. Cualquier jugador puede coger 1, 2 o 3 pequines en su turno. El jugador que coja el último penique pierde. Prediga qué jugador nunca perderá.

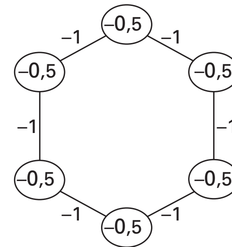
39. Suponga que nuestro trabajo consiste en supervisar la carga de dos camiones, cada uno de los cuales puede transportar como máximo catorce toneladas. La carga es un conjunto de contenedores cuyo peso total es de veintiocho toneladas, pero cuyo peso individual varía de un contenedor a otro. El peso de cada contenedor está marcado en un lateral. ¿Qué heurístico utilizaría para repartir los contenedores entre los dos camiones?

40. ¿Cuáles de los siguientes enunciados son ejemplos de meta-razonamiento?
   a. Hace mucho tiempo que se ha ido, así que debe de estar lejos.
   b. Como usualmente tomo la decisión incorrecta y las dos últimas decisiones que tomé eran correctas, voy a invertir mi siguiente decisión.
   c. Estoy cansado, así que probablemente no pienso con claridad.
   d. Estoy cansado, así que voy a echarme una siesta.

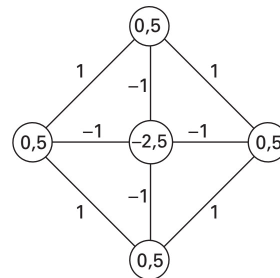
41. ¿Es posible desarrollar un sistema de inteligencia artificial que pueda escribir un algoritmo para resolver un problema?

42. a. ¿En qué se asemejan el aprendizaje por imitación y el aprendizaje mediante entrenamiento supervisado?
   b. ¿En qué se diferencian el aprendizaje por imitación y el aprendizaje mediante entrenamiento supervisado?

43. El siguiente diagrama representa una red neuronal artificial para una memoria asociativa, tal como se explica en la Sección 11.5. ¿Qué patrón se asocia con cualquier patrón en el que solo estén excitadas dos neuronas que estén separadas por solo una neurona? ¿Qué sucede si se inicializa la red con todas sus neuronas inhibidas?



44. El siguiente diagrama representa una red neuronal artificial para una memoria asociativa, tal como se explica en la Sección 11.5. ¿Qué configuración estable asocia con cualquier patrón inicial en el que al menos tres de las neuronas del perímetro estén excitadas y la neurona central esté inhibida? ¿Qué sucedería si se le proporcionara un patrón inicial en el que solo estuvieran excitadas dos neuronas del perímetro situadas en posición opuesta la una a la otra?



45. Diseñe una red neuronal artificial para una memoria asociativa (como se explica en la Sección 11.5) que conste de una matriz rectangular de neuronas que trate de derivar hacia patrones estables en los que solo esté excitada una única columna vertical de neuronas.

46. Ajuste los pesos y valores de umbral de la red neuronal artificial de la Figura 11.18 para que su salida sea 1 cuando ambas entradas coincidan (ambas sean 0 o ambas sean 1) y 0 cuando las entradas sean diferentes (una de ellas sea 0, mientras que la otra sea 1).

47. Dibuje un diagrama similar al de la Figura 11.5 que represente el proceso de simplificar la expresión algebraica $7x + 3 = 3x - 5$ para obtener la expresión $x = -2$.

48. Amplíe su respuesta al problema anterior para mostrar otras rutas que un sistema de control podría explorar a la hora de tratar de resolver el problema.

49. Dibuje un diagrama similar al de la Figura 11.5 que represente el proceso de razonamiento implicado a la hora de concluir que "Poli puede volar" a partir de los hechos iniciales "Poli es un loro", "Un loro es un pájaro" y "Todos los pájaros pueden volar".

50. A pesar de lo que se dice en el enunciado del problema anterior, algunos pájaros, como por ejemplo el avestruz o un pájaro que tenga un ala rota, no pueden volar. Sin embargo, no sería razonable construir un sistema de razonamiento deductivo en el que se enumeraran explícitamente todas las excepciones al enunciado "Todos los pájaros pueden volar". ¿Cómo decidimos entonces los seres humanos si un pájaro concreto puede o no volar?

51. Explique cómo depende del contexto el significado de la frase "He leído la nueva ley impositiva".

52. Describa cómo podría expresarse en forma de sistema de producción el problema de viajar de una ciudad a otra. ¿Cuáles son los estados? ¿Cuáles son las producciones?

53. Suponga que debemos realizar tres tareas, A, B y C, y que pueden llevarse a cabo en cualquier orden (pero no simultáneamente). Describa cómo puede expresarse en forma de sistema de producción este problema y dibuje su grafo de estados.

54. ¿Cuáles son los componentes que definen un sistema experto?

55. a. Si la notación (i, j), donde i y j son enteros positivos, se utiliza para decir que " si la entrada situada en la posición i-ésima de la lista es mayor que la entrada de la posición j-ésima, intercambiar las dos entradas", ¿cuál de las siguientes dos secuencias será más adecuada a la hora de ordenar una lista de longitud igual a tres?

(1, 3) (3, 2)

(1, 2) (2, 3) (1, 2)

b. Observe que al representar secuencias de intercambios de esta manera, las secuencias pueden descomponerse en subsecuencias que luego pueden reconectarse para formar nuevas secuencias. Utilice esta técnica para describir un algoritmo genético que permita desarrollar un programa que ordene listas de longitud igual a diez.

56. Suponga que cada miembro de un grupo de robots debe equiparse con una pareja de sensores. Cada sensor puede detectar un objeto situado delante de él y tiene un alcance de dos metros. Cada robot tiene la forma de una papelera redonda y puede moverse en cualquier dirección. Diseñe una secuencia de experimentos para determinar dónde hay que colocar los sensores para obtener un robot que empuje correctamente una pelota de baloncesto a lo largo de una línea recta. ¿Cómo compararía su secuencia de experimentos con un sistema evolutivo?

57. ¿Puede un sistema de inteligencia artificial predecir la situación financiera de una bolsa de valores? En caso afirmativo, ¿cómo de complejo sería el sistema? ¿Qué distintas restricciones deberían tenerse en consideración?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. ¿Hasta qué grado deben ser considerados responsables de la forma en que se utilicen los resultados de su trabajo los investigadores en energía nuclear, en ingeniería genética y en inteligencia artificial? ¿Son los científicos responsables de los conocimientos que adquirimos gracias a sus investigaciones? ¿Qué pasa si esos conocimientos son una consecuencia inesperada de sus investigaciones?
2. ¿Cómo diferenciaría entre la inteligencia y la inteligencia simulada? ¿Existe alguna diferencia?
3. Suponga que un sistema experto computerizado para aplicaciones médicas se labra una gran reputación dentro de la comunidad médica por proporcionar consejos correctos. ¿Hasta qué punto debería un médico permitir que ese sistema alterara sus decisiones relativas a los tratamientos de los pacientes? Si el médico aplica un tratamiento contrario al propuesto por el sistema experto y resulta que el sistema tenía razón, ¿se puede acusar de mala praxis a ese médico? En general, si un sistema experto llega a ser muy conocido dentro de un campo, ¿hasta qué grado podría amenazar, en lugar de mejorar, la capacidad de los expertos humanos para llegar a sus propias conclusiones?
4. Muchas personas argumentan que las acciones de una computadora son simplemente consecuencias del modo en que fue programada, por lo que la computadora no puede poseer libre albedrío. Por ello, una computadora no puede ser considerada responsable de sus acciones. ¿Cree que la mente humana es una computadora? ¿Están los humanos preprogramados desde su nacimiento? ¿Estamos los humanos programados por nuestro entorno? ¿Somos los humanos responsables de nuestras acciones?
5. ¿Existen caminos en los que la ciencia no debería adentrarse aún cuando sea capaz de hacerlo? Por ejemplo, si llegara a ser posible construir una máquina con capacidades de percepción y razonamiento comparables a las de los seres humanos, ¿sería apropiada la construcción de una de tales máquinas? ¿Qué problemas plantearía la existencia de una de esas máquinas? ¿Podría citar algunos de los problemas planteados por los avances en otros campos de la ciencia?
6. La historia está llena de casos en los que el trabajo de científicos y artistas se vió afectado por influencias políticas, religiosas u otras influencias sociales presentes en aquel momento. ¿En qué forma están ese tipo de cuestiones afectando a las actuales tareas de investigación científica? ¿Y qué sucede con las Ciencias de la computación en particular?
7. Muchos países actuales aceptan al menos una parte de responsabilidad a la hora de ayudar a formar de nuevo a aquellas personas cuyos trabajos han

dejado de ser necesarios a causa de los avances de la tecnología. ¿Qué debería/podría hacer la sociedad a medida que la tecnología haga que cada vez más capacidades de los seres humanos sean innecesarias?

8. Suponga que recibimos una factura generada por computadora por un importe de 0,00 euros. ¿Qué deberíamos hacer? Suponga que no hacemos nada y que 30 días después recibimos un segundo aviso de que van a cargar 0,00 euros en nuestra cuenta. ¿Qué deberíamos hacer? Suponga que no hacemos nada y que 30 días después recibimos otro aviso de que se han cargado 0,00 euros en nuestra cuenta junto con una nota que indica que a menos que la factura se pague con prontitud se emprenderán acciones legales. ¿Quién se supone que está a cargo de las cosas?

9. ¿Ha sentido la tentación alguna vez de asociar una personalidad con su computadora personal? ¿Cree que hay veces en las que la computadora parece vengativa o tozuda? ¿Alguna vez se ha enfadado con su computadora? ¿Cuál es la diferencia entre enfadarse ***con*** una computadora y enfadarse ***como resultado de lo que la computadora hace***? ¿Cree que su computadora se ha enfadado alguna vez con usted? ¿Ha experimentado alguna vez una relación similar con otros objetos, como vehículos, televisiones o plumas de escribir?

10. Basándose en sus respuestas a la Cuestión 9, ¿hasta qué grado cree que los seres humanos deseamos asociar el comportamiento de una entidad con la presencia de inteligencia y de consciencia? ¿Hasta qué grado deberían los humanos efectuar ese tipo de asociaciones? ¿Es posible para una entidad inteligente revelar su inteligencia de alguna otra manera que a través de su comportamiento?

11. Muchos expertos piensan que la capacidad de pasar el test de Turing no implica que una máquina sea inteligente. Uno de los argumentos utilizados es que el comportamiento inteligente no implica, por sí mismo, inteligencia. A pesar de ello, la teoría de la evolución está basada en la supervivencia del más adaptado, que es un test basado en el comportamiento. ¿Implica la teoría de la evolución que el comportamiento inteligente es un predecesor de la inteligencia? ¿Implicaría la capacidad de pasar el test de Turing que las máquinas están en camino de volverse inteligentes?

12. El tratamiento médico ha avanzado hasta tal punto que numerosas partes del cuerpo humano pueden ahora sustituirse por partes artificiales o por partes obtenidas de donantes humanos. Ahora es concebible que esto pueda llegar a incluir algún día partes del cerebro. ¿Qué problemas éticos plantearían esas capacidades? Si se sustituyeran las neuronas de un paciente por neuronas artificiales de una en una, ¿seguiría ese paciente siendo la misma persona? ¿Llegaría alguna vez a notar alguna diferencia ese paciente? ¿Seguiría siendo humano ese paciente?

13. Un GPS de un automóvil proporciona una voz amigable que informa al conductor de los giros próximos y de otras acciones. En caso de que el conductor cometa un error, el GPS realizará ajustes automáticamente y proporcionará instrucciones para volver a la ruta prevista, sin mostrar ninguna emoción indebida. ¿Cree que un GPS reduce el estrés de un conductor a la

hora de conducir hasta un nuevo destino? ¿En qué formas contribuye un GPS al estrés?

14. Suponga que su teléfono inteligente proporcionara traducción de idiomas voz a voz. ¿Se sentiría cómodo utilizando esta funcionalidad? ¿Confiaría en que fuera capaz de transmitir el significado correcto de sus palabras? ¿Provocaría en usted algún tipo de preocupación?

Lecturas adicionales

Banzhaf, W., P. Nordin, R. E. Deller y F. D. Francone. *Genetic Programming: An Introduction.* San Francisco, CA: Morgan Kaufmann, 1998.

Lu, J. y J. Wu. *Multi-Agent Robotic Systems*. Boca Raton, FL: CRC Press, 2001.

Luger, G. F. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving,* 5ª ed. Boston, MA: Addison-Wesley, 2005.

Mitchell, M. *An Introduction to Genetic Algorithms.* Cambridge, MA: MIT Press, 1998.

Negnevitsky, M. *Artificial Intelligence: A Guide to Intelligent Systems,* 2ª ed. Boston, MA: Addison-Wesley, 2005.

Nilsson, N. *Artificial Intelligence: A New Synthesis.* San Francisco, CA: Morgan Kaufmann, 1998.

Nolfi, S. y D. Floreano. *Evolutionary Robotics*. Cambridge, MA: MIT Press, 2000.

Rumelhart, D. E. y J. L. McClelland. *Parallel Distributed Processing.* Cambridge, MA: MIT Press, 1986.

Russell, S. y P. Norvig. *Artificial Intelligence: A Modern Approach,* 3ª ed. Upper Saddle River, NJ: Prentice-Hall, 2009.

Shapiro, L. G. y G. C. Stockman. *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 2001.

Shieber, S. *The Turing Test.* Cambridge, MA: MIT Press, 2004.

Weizenbaum, J. *Computer Power and Human Reason.* Nueva York: W. H. Freeman, 1979.

CAPÍTULO 12

Teoría de la computación

En este capítulo vamos a considerar los fundamentos teóricos de las Ciencias de la computación. En cierto sentido, es el material presentado en este capítulo el que proporciona a las Ciencias de la computación la consideración de verdadera ciencia. Aunque de una naturaleza algo abstracta, este cuerpo de conocimientos tiene muchas aplicaciones de carácter eminentemente práctico. En particular, exploraremos sus implicaciones en relación con la potencia de los lenguajes de programación y veremos cómo conducen a la definición de un sistema de cifrado de clave pública ampliamente utilizado para las comunicaciones a través de Internet.

**Las secciones marcadas con asterisco se sugieren como secciones opcionales.*

En este capítulo vamos a considerar una serie de cuestiones relativas a lo que las computadoras pueden o no pueden hacer. Veremos cómo hay unas máquinas simples, denominadas máquinas de Turing, que se utilizan para identificar la frontera entre problemas resolubles mediante máquinas y problemas que no lo son. Analizaremos un problema concreto, conocido con el nombre de problema de la detención, cuya solución escapa a la capacidad de los sistemas algorítmicos y cae, por tanto, fuera de la capacidad de las computadoras tanto actuales como futuras. Además, veremos que incluso entre el conjunto de problemas resolubles mediante máquinas, existen problemas cuya solución es tan compleja que podemos considerarlos irresolubles desde el punto de vista práctico. Cerraremos el capítulo analizando cómo esos conocimientos en el área de la complejidad pueden utilizarse para construir un sistema de cifrado de clave pública.

12.1 Funciones y su computabilidad

Nuestro objetivo en este capítulo es investigar las capacidades de las computadoras. Queremos entender lo que las máquinas pueden o no pueden hacer y qué características se requieren para que las máquinas puedan desarrollar todo su potencial. Comenzaremos con el concepto de computación de funciones.

Una **función** en su sentido matemático es una correspondencia entre un conjunto de posibles valores de entrada y un conjunto de valores de salida, de modo que a cada una de las posibles entradas se le asigna una única salida. Un ejemplo sería la función que convierte las medidas en yardas a metros. A cada medida en yardas, la función le asigna un valor que es el que se obtendría si se midiera esa misma distancia en metros. Otro ejemplo, que podríamos denominar función de ordenación, asigna a cada lista de entrada compuesta por valores numéricos una lista de salida cuyas entradas son las mismas que las de la lista de entrada pero dispuestas en orden creciente. Otro ejemplo sería la función suma cuyas entradas son parejas de valores y cuyas salidas son los valores que representan la suma de cada pareja de entrada.

El proceso de determinar el valor concreto de salida que una cierta función asigna a una entrada determinada se denomina ***computación de la función.*** La habilidad de calcular funciones es importante, porque si somos capaces de

Teoría de funciones recursivas

Nada molesta más a la naturaleza humana que el que nos digan que no se puede hacer algo. Una vez que los investigadores comenzaron a identificar problemas que son irresolubles, en el sentido de que no disponen de solución algorítmica, otras personas comenzaron a estudiar esos problemas intentando comprender su complejidad. Hoy día, esta área de investigación es una de las partes principales de un tema conocido con el nombre de teoría de funciones recursivas, y hemos aprendido muchas cosas acerca de estos problemas superdifíciles. De hecho, al igual que los matemáticos han desarrollado sistemas de numeración que permiten revelar niveles "cuantitativos" más allá del infinito, los teóricos de las funciones recursivas han descubierto múltiples niveles de complejidad dentro de problemas que yacen mucho más allá de las capacidades de los algoritmos.

resolver los problemas es por medio de ese proceso de computar funciones. Para resolver un problema de suma debemos calcular la función suma, para ordenar una lista debemos computar la función de ordenación. Por ello, una de las tareas fundamentales de las Ciencias de la computación consiste en encontrar técnicas para computar las funciones que subyacen a los problemas que deseemos resolver.

Considere, por ejemplo, un sistema en el que las entradas y salidas de una función puedan ser predeterminadas y anotadas en una tabla. Cada vez que se necesite conocer la salida de la función, basta con buscar las entradas dadas en la tabla, donde podremos ver cuál es la salida requerida. Por tanto, la computación de esa función se reduciría al problema de buscar en la tabla. Dichos sistemas son cómodos, pero su potencia es limitada, porque muchas funciones no pueden representarse de manera completa en una tabla. Un ejemplo es el que se muestra en la Figura 12.1, que es un intento de visualizar la función que convierte las medidas en yardas en sus medidas equivalentes en metros. Puesto que no existe ningún límite a la lista de posibles parejas de entrada/salida, la tabla siempre estará obligatoriamente incompleta.

Una técnica más potente para el cálculo de funciones consiste en seguir las directrices proporcionadas por una fórmula algebraica, en lugar de tratar de visualizar en una tabla todas las posibles combinaciones de entrada/salida. Por ejemplo, podemos utilizar la fórmula algebraica

$$V = P(1 + r)^n$$

para describir cómo calcular el valor de una inversión P después de aplicar una tasa de interés anual compuesto de r durante n años.

Pero la potencia expresiva de las fórmulas algebraicas también tiene sus limitaciones. Existen funciones cuyas relaciones de entrada/salida son demasiado complejas como para poderlas describir mediante manipulaciones algebraicas. Como ejemplos podríamos citar las funciones trigonométricas como el seno o el coseno. Si nos obligan a calcular el seno de 38 grados, podemos dibujar el triángulo apropiado, medir sus lados y calcular la relación deseada, un proceso que no se puede expresar en términos de manipulaciones algebraicas del valor 38. Nuestra calculadora de bolsillo también tiene que realizar malabarismos para resolver la tarea de calcular el seno de 38 grados. En realidad, está

Figura 12.1 Un intento de mostrar la función que convierte las medidas en yardas en sus medidas equivalentes en metros.

Yardas (entrada)	Metros (salida)
1	0,9144
2	1,8288
3	2,7432
4	3,6576
5	4,5720
.	.
.	.
.	.

obligada a aplicar técnicas matemáticas bastante sofisticadas con el fin de obtener una muy buena aproximación al seno de 38 grados y es esa aproximación lo que nos da como respuesta.

Podemos ver, por tanto, que a medida que consideramos funciones de complejidad creciente, nos vemos obligados a aplicar técnicas más potentes para computarlas. La cuestión que nos planteamos es si siempre podremos encontrar un sistema para computar las funciones, independientemente de su complejidad. La respuesta a esta pregunta es no. Un sorprendente resultado de las matemáticas es que existen funciones tan complejas que no existe ningún proceso paso a paso bien definido para determinar sus salidas a partir de sus valores de entrada. Por ello, la computación de esas funciones cae más allá de las capacidades de cualquier sistema algorítmico. De esas funciones decimos que son no computables, mientras que las funciones cuyos valores de salida se pueden determinar algorítmicamente a partir de sus valores de entrada se denominan **computables**.

La distinción entre funciones computables y no computables es muy importante en las Ciencias de la computación. Puesto que las máquinas solo pueden realizar tareas descritas mediante algoritmos, el estudio de las funciones computables es el estudio de los límites de las capacidades de las máquinas. Si podemos identificar capacidades que permitan a una máquina computar el conjunto completo de funciones computables y luego construimos máquinas con estas capacidades, podremos estar seguros de que las máquinas que construyamos son lo más potentes posible. De la misma forma, si descubrimos que la solución a un problema requiere la computación de una función no computable, podemos concluir que la solución de ese problema cae más allá de las capacidades de las máquinas.

Cuestiones y ejercicios

1. Identifique algunas funciones que puedan representarse completamente en forma tabular.
2. Identifique algunas funciones cuyas salidas puedan describirse en forma de una expresión algebraica en la que se incluyan sus entradas.
3. Identifique una función que no pueda describirse en términos de una fórmula algebraica. ¿Es esa función, de todos modos, computable?
4. Los antiguos matemáticos griegos utilizaban una regla y un compás para dibujar formas geométricas. Desarrollaron técnicas para hallar el punto medio de una recta, construir un ángulo recto y dibujar un triángulo equilátero. Sin embargo, ¿podría citar algunas "computaciones" que su "sistema computacional" no pudiera realizar?

12.2 Máquinas de Turing

En un intento de comprender las capacidades y limitaciones de las máquinas, muchos investigadores han propuesto y estudiado diversos dispositivos computacionales. Uno de ellos es la máquina de Turing, que fue propuesta por Alan

Orígenes de las máquinas de Turing

Alan Turing desarrolló el concepto de máquina de Turing en la década de 1930, mucho antes de que la tecnología fuera capaz de proporcionarnos las técnicas que conocemos hoy día. De hecho, la visión de Turing era la de un ser humano realizando los cálculos con lápiz y papel. El objetivo de Turing era el de proporcionar un modelo con el que estudiar los límites de los "procesos computacionales". Esto fue poco después de la publicación en 1931 del famoso artículo de Gödel en el que se exponían las limitaciones de los sistemas computacionales, y grandes esfuerzos de investigación se dirigieron a intentar comprender esas limitaciones. En el mismo año en que Turing presentó su modelo (1936), Emil Post presentó otro modelo (ahora conocido con el nombre de sistemas de producción de Post) que ha demostrado tener las mismas capacidades que el de Turing. Como testimonio de la visión de estos primeros investigadores, sus primeros modelos de sistemas computacionales (como las máquinas de Turing y los sistema de producción de Post) siguen sirviendo como herramientas muy valiosas en las investigaciones efectuadas en el campo de las Ciencias de la computación.

M. Turing en 1936 y continúa utilizándose hoy día para estudiar la potencia de los procesos algorítmicos.

Fundamentos de la máquina de Turing

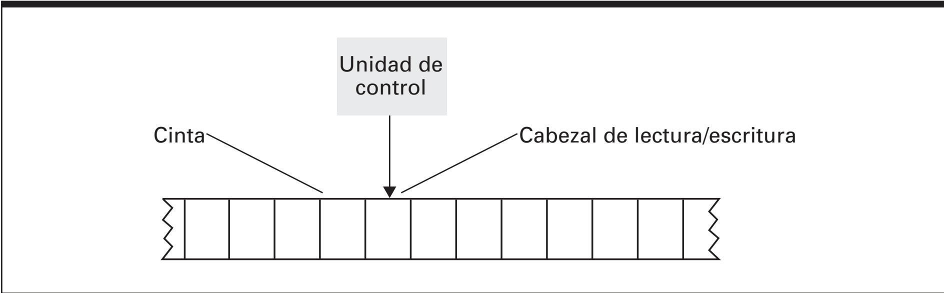
Una **máquina de Turing** está compuesta por una unidad de control que puede leer y escribir símbolos en una cinta por medio de un cabezal de lectura/escritura (Figura 12.2). La cinta se extiende indefinidamente por ambos extremos y está dividida en casillas, cada una de las cuales puede contener un símbolo de entre un conjunto finito de ellos. Este conjunto se denomina alfabeto de la máquina.

En todo momento durante los cálculos realizados con una máquina de Turing, la máquina debe estar en una de entre un número finito de condiciones, denominados estados. La computación efectuada por una máquina de Turing comienza en un estado especial denominado estado inicial y termina cuando la máquina alcanza otro estado especial conocido con el nombre de estado de detención.

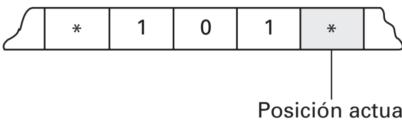
La computación efectuada por una máquina de Turing consiste en una secuencia de pasos ejecutados por la unidad de control de la máquina. Cada paso consiste en observar el símbolo escrito en la casilla actual de la cinta (la que está viendo el cabezal de lectura/escritura), escribir un símbolo en esa casilla, mover posiblemente el cabezal de lectura/escritura una casilla hacia la izquierda o hacia la derecha y luego cambiar de estado. La acción concreta que hay que realizar está determinada por un programa que le dice a la unidad de control lo que hay que hacer basándose en el estado de la máquina y en el contenido de la casilla actual de la cinta.

Veamos un ejemplo de una máquina de Turing específica. Con este objetivo, vamos a representar la cinta de la máquina como una banda horizontal dividida en casillas en las que podemos anotar símbolos pertenecientes al alfabeto de la máquina. Indicaremos la posición actual del cabezal de lectura/escri-

Figure 12.2 Los componentes de una máquina de Turing.



tura de la máquina colocando una etiqueta alusiva debajo de la casilla actual de la cinta. El alfabeto para nuestro ejemplo estará compuesto por los símbolos 0, 1 y *. La cinta de nuestra máquina podría tener la siguiente apariencia:



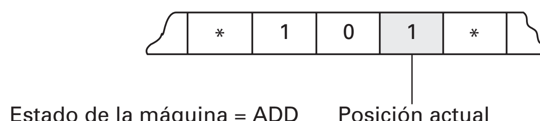
Interpretando una cadena de símbolos de la cinta como si representara una serie de números binarios separados por asteriscos, podemos reconocer que esta cinta concreta contiene el valor 5. Nuestra máquina de Turing está diseñada para incrementar en una unidad ese valor de la cinta. Para ser más precisos, vamos a suponer que en la posición inicial hay un asterisco que marca el extremo derecho de una cadena de 0s y 1s, y que la cinta debe modificar el patrón de bits situado a su izquierda, para representar el siguiente entero.

Los estados de nuestra máquina serán START (estado inicial) , ADD (sumar), CARRY (acarreo), OVERFLOW (desbordamiento), RETURN (volver) y HALT (detención). Las acciones correspondientes a cada uno de estos estados y el contenido de la casilla correspondiente se describen en la tabla de la Figura 12.3. Vamos a suponer que la máquina comienza siempre en el estado START.

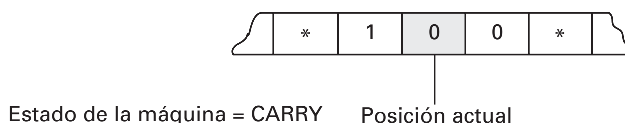
Figura 12.3 Una máquina de Turing para incrementar un valor.

Estado actual	Contenido de la casilla actual	Valor que hay que escribir	Dirección de movimiento	Estado al que pasar
START	*	*	Izquierda	ADD
ADD	0	1	Derecha	RETURN
ADD	1	0	Izquierda	CARRY
ADD	*	*	Derecha	HALT
CARRY	0	1	Derecha	RETURN
CARRY	1	0	Izquierda	CARRY
CARRY	*	1	Izquierda	OVERFLOW
OVERFLOW	(Ignorado)	*	Derecha	RETURN
RETURN	0	0	Derecha	RETURN
RETURN	1	1	Derecha	RETURN
RETURN	*	*	Sin movimiento	HALT

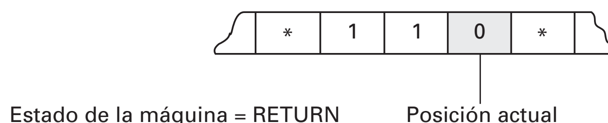
Vamos a aplicar esta máquina a la cinta que hemos mostrado anteriormente que contenía el valor 5. Observe que, cuando nos encontramos con el estado START con la casilla actual conteniendo * (como es nuestro caso), la tabla nos dice que hay que reescribir el *, mover el cabezal de lectura/escritura una casilla hacia la izquierda y entrar en el estado ADD. Habiéndose hecho esto, la máquina se puede describir como sigue:



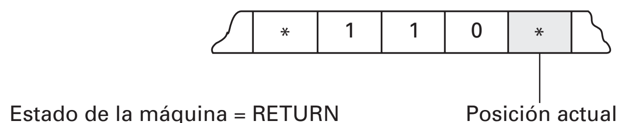
Para continuar, examinamos la tabla para ver qué es lo que tenemos que hacer cuando estamos en estado ADD y la casilla actual contiene un 1. La tabla dice que hay que sustituir el 1 de la casilla actual por un 0, mover el cabezal de lectura/escritura una casilla hacia la izquierda y entrar en el estado CARRY. Por tanto, la configuración de la máquina pasará a ser:



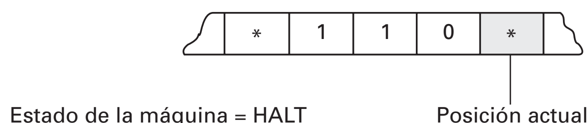
Consultamos de nuevo la tabla para ver qué hay que hacer a continuación y vemos que cuando nos encontramos en el estado CARRY y la casilla actual contiene un 0, tenemos que sustituir el 0 por un 1, mover el cabezal de lectura/escritura una casilla hacia la derecha y entrar en el estado RETURN. Después de esto, la configuración de la máquina será la siguiente:



A partir de aquí, la tabla nos dice que debemos continuar sustituyendo el 0 de la casilla actual por otro 0, mover el cabezal de lectura/escritura una casilla hacia la derecha y permanecer en el estado RETURN. En consecuencia, nos encontraremos nuestra máquina en la siguiente condición:



En este punto, vemos que la tabla nos dice que hay que reescribir el asterisco en la casilla actual y entrar en el estado HALT. La máquina se parará así en la siguiente configuración (los símbolos de la cinta representan ahora el valor 6, como deseábamos):



La tesis de Church-Turing

La máquina de Turing del ejemplo anterior se puede utilizar para calcular la función conocida con el nombre de función sucesora, que asigna a cada entero no negativo n que se le proporcione como valor de entrada, un valor de salida igual a $n + 1$. Simplemente necesitamos colocar el valor de entrada en formato binario en la cinta de la máquina, operar la máquina hasta que se detenga y luego leer el valor de salida de la cinta. Toda función que pueda computarse de esta forma mediante una máquina de Turing se dice que es **computable según Turing**.

La conjetura de Turing era que el conjunto de las funciones computables según Turing coincide con el conjunto de todas las funciones computables. En otras palabras, conjeturó que la potencia computacional de la máquina de Turing incluye la de cualquier sistema algorítmico o, lo que es lo mismo, que (a diferencia de otros enfoques, como los de las tablas o las fórmulas algebraicas) el concepto de máquina de Turing proporciona un contexto con el que expresar las soluciones para todas las funciones computables. Hoy día, esta conjetura se denomina a menudo **tesis de Church–Turing**, en referencia a las contribuciones hechas tanto por Alan Turing como por Alonzo Church. Desde el trabajo inicial de Turing, se han recopilado muchas pruebas para apoyar esta tesis, y hoy día la tesis de Church–Turing es ampliamente aceptada. Es decir, se considera que el conjunto de las funciones computables y el conjunto de las funciones computables según Turing coinciden.

La importancia de esta conjetura es que nos proporciona información acerca de las capacidades y limitaciones de las máquinas de computación. De forma más precisa, establece las capacidades de las máquinas de Turing como un estándar con el que comparar la potencia de otros sistema computacionales. Si un sistema computacional es capaz de calcular todas las funciones computables según Turing, se considera que su potencia es igual a la máxima posible que cualquier sistema computacional puede ofrecer.

Cuestiones y ejercicios

1. Aplique la máquina de Turing descrita en esta sección (Figura 12.3), comenzando a partir del siguiente estado inicial:



2. Describa una máquina de Turing que sustituya una cadena de 0s y 1s por un único 0.
3. Describa una máquina de Turing que decremente en una unidad el valor de la cinta si este es mayor que cero o deje el valor como está si es igual a cero.
4. Identifique una situación cotidiana en la que se realice algún tipo de cálculo. ¿En qué sentido es análoga esa situación a una máquina de Turing?

5. Describa una máquina de Turing que termine por detenerse para algunas entradas pero que nunca se detenga para otras.

12.3 Lenguajes de programación universales

En el Capítulo 6 hemos estudiado diversas características que podemos encontrar en los lenguajes de programación de alto nivel. En esta sección vamos a aplicar nuestros conocimientos acerca de la computabilidad para determinar cuáles de esas características son realmente necesarias. Veremos que la mayoría de las características de los lenguajes de alto nivel actuales simplemente los hacen más cómodos sin por ello contribuir a aumentar la potencia fundamental del lenguaje.

Nuestro enfoque consistirá en describir un lenguaje de programación simple de carácter imperativo que sea lo suficientemente rico como para permitirnos expresar programas que sirvan para calcular todas las funciones computables según Turing (y por tanto todas las funciones computables). Por tanto, si un futuro programador comprueba que un problema no puede resolverse utilizando este lenguaje, la razón no será un fallo del lenguaje, sino más bien que no existe ningún algoritmo para resolver el problema. Un lenguaje de programación con esta propiedad se denomina **lenguaje de programación universal**.

Puede que el lector encuentre sorprendente el hecho de que un lenguaje universal no necesite ser complejo. De hecho, el lenguaje que vamos a presentar es bastante simple. Lo llamaremos Bare Bones (N. del T.: que podría traducirse como lenguaje mínimo) porque aísla el conjunto mínimo de requisitos que es necesario imponer a un lenguaje de programación universal.

El lenguaje de Bare Bones

Comenzamos nuestra presentación de Bare Bones considerando las sentencias declarativas que podemos encontrar en otros lenguajes de programación. Estas sentencias permiten a los programadores darse el lujo de pensar en términos de estructuras de datos y de tipos de datos (tales como matrices de valores numéricos y cadenas de caracteres alfabéticos), aunque la propia máquina se limite a manipular los patrones de bits sin tener ningún conocimiento de qué es lo que esos patrones representan. Antes de presentarla a una máquina para que la ejecute, una sentencia de alto nivel que trate con estructuras y tipos de datos elaborados debe ser traducida a instrucciones de lenguaje máquina que manipulen patrones de bits, con el fin de simular las acciones solicitadas.

Por comodidad, podemos interpretar esos patrones como valores numéricos representados en notación binaria. Por tanto, todos los cálculos realizados por una computadora podrían expresarse como cálculos numéricos con enteros no negativos (como vemos todo depende de cómo se miren las cosas). Además, los lenguajes de programación se podrían simplificar exigiendo a los programadores que expresaran los algoritmos en estos términos (aunque esto obligaría al programador a realizar una tediosa labor).

Puesto que nuestro objetivo al desarrollar Bare Bones es desarrollar el lenguaje más simple posible, vamos a seguir precisamente este camino: consideraremos que todas las variables de Bare Bones representan patrones de bits que, por comodidad, interpretaremos como enteros no negativos en notación binaria. Por tanto, si una variable tiene actualmente asignado el patrón 10 diremos que contiene el valor dos, mientras que si tiene asignado el patrón 101 diremos que contiene el valor cinco.

Utilizando este convenio, todas las variables de un programa Bare Bones son del mismo tipo, por lo que el lenguaje no necesita sentencias declarativas para describir los nombres de las variables y sus propiedades asociadas. Al emplear Bare Bones, un programador puede simplemente comenzar a utilizar un nuevo nombre de variable en el momento que sea necesario, asumiendo que hace referencia a un patrón de bits interpretado como un entero no negativo.

Por supuesto, un traductor para nuestro lenguaje Bare Bones deberá ser capaz de distinguir los nombres de variables de otros términos. Esto se hace diseñando la sintaxis de Bare Bones de modo que el papel de cualquier término pueda identificarse a partir de la propia sintaxis. Con este objeto, decidimos que los nombres de las variables deben comenzar con una letra del alfabeto inglés, que puede ir seguida por cualquier combinación de letras y dígitos (0 a 9). Por tanto, las cadenas `XYZ`, `B747`, `abcdefghi` y `X5Y` pueden utilizarse como nombres de variable, mientras que `2G5`, `%o` y `x.y` no.

Consideremos ahora las sentencias procedimentales de Bare Bones. Existen tres sentencias de asignación y una estructura de control que representa un bucle. El lenguaje es un lenguaje de formato libre, por lo que cada sentencia termina con un punto y coma, lo que facilita al traductor la tarea de separar las sentencias que aparezcan en la misma línea. Sin embargo, nosotros adoptaremos la política de escribir una única sentencia por línea con el fin de mejorar la legibilidad de los programas.

Cada una de las tres sentencias de asignación solicita que se modifique el contenido de la variable identificada en la sentencia. La primera de ellas nos permite asociar el valor cero con una variable. Su sintaxis es

```
clear nombre;
```

donde *nombre* puede ser cualquier nombre de variable.

Las otras sentencias de asignación son esencialmente opuestas entre sí:

```
incr nombre;
```

y

```
decr nombre;
```

De nuevo, *nombre* representa cualquier nombre de variable. La primera de estas sentencias hace que el valor asociado con la variable identificada se incremente en una unidad. Por tanto, si la variable `Y` tuviera asignado el valor cinco antes de ejecutar la sentencia

```
incr Y;
```

entonces el valor asignado a `Y` después de ejecutada la sentencia sería seis.

Por el contrario, la sentencia `decr` se utiliza para decrementar en una unidad el valor asociado con la variable identificada. Una excepción es cuando la variable ya tiene asociado el valor cero, en cuyo caso la sentencia no modifica

el valor. Por tanto, si el valor asociado con `Y` es cinco antes de ejecutar la sentencia

```
decr Y;
```

entonces el valor asociado con `Y` después de ejecutar la sentencia será cuatro. Sin embargo, si el valor de `Y` fuera cero antes de ejecutar la sentencia, ese valor continuaría siendo cero después de ejecutarla.

Bare Bones solo proporciona una estructura de control representada por una pareja de sentencias `while-end`. La secuencia de sentencias

```
while nombre not 0 do;
  .
  .
  .
end;
```

(donde *nombre* representa cualquier nombre de variable) hace que cualquier sentencia o secuencia de sentencias colocada entre las sentencias `while` y `end` se repita mientras que el valor de la variable *nombre* sea distinto de cero. Para ser más precisos, cuando nos encontramos una estructura `while-end` durante la ejecución del programa, se comprueba en primer lugar si el valor de la variable identificada es cero. Si lo es, nos saltamos la estructura y la ejecución continúa en la instrucción que se encuentra después de la instrucción `end`. Sin embargo, si el valor de la variable es distinto de cero, se ejecuta la secuencia de sentencias contenida dentro de la estructura `while-end` y se devuelve el control a la sentencia `while`, donde se vuelve a comparar el valor con cero. Observe que será parcialmente responsabilidad del programador toda la complejidad asociada con el control del bucle, ya que la sentencia deberá hacer explícitamente que se modifique el valor de la variable dentro del cuerpo del bucle con el fin de evitar un bucle infinito. Por ejemplo, la secuencia

```
incr X;
while X not 0 do;
  incr Z;
end;
```

da como resultado un proceso infinito, porque una vez alcanzada la sentencia `while`, el valor asociado con `X` nunca puede ser cero, mientras que la secuencia

```
clear Z;
while X not 0 do;
  incr Z;
  decr X;
end;
```

llegará a terminar, con el efecto de transferir a la variable `Z` el valor inicialmente asociado con `X`.

Observe que las sentencias `while` y `end` deben aparecer emparejadas, y que la sentencia `while` debe aparecer primero. Sin embargo, una pareja de sentencias `while-end` puede aparecer dentro de las sentencias ejecutadas por otra pareja de sentencias `while-end`. En dicho caso, el emparejamiento de sentencias `while` y `end` se lleva a cabo analizando el programa en su forma escrita, de principio a fin, y asociando cada sentencia `end` con la sentencia `while` prece-

dente más próxima que todavía no haya sido emparejada. Aunque no es sintácticamente necesario, utilizaremos a menudo el sangrado para mejorar la legibilidad de tales estructuras.

Como ejemplo final, la secuencia de sentencias de la Figura 12.4 hace que se asigne a `Z` el producto de los valores asociados con `X` e `Y`, aunque tiene el efecto colateral de destruir cualquier valor distinto de cero que pudiera haber estado asociado con `X`. (La estructura `while-end` controlada por la variable `W` tiene el efecto de restaurar el valor original de `Y`.)

Programación en Bare Bones

Recuerde que nuestro objetivo al presentar el lenguaje Bare Bones es investigar qué cosas son posibles; no qué cosas resultan prácticas. Bare Bones sería terrible de utilizar en el entorno de una aplicación real. Por otro lado, pronto veremos que este lenguaje tan simple satisface completamente nuestro objetivo de disponer de un lenguaje de programación universal mínimo. Por ahora, simplemente vamos a mostrar cómo puede utilizarse Bare Bones para expresar algunas operaciones elementales.

En primer lugar, observemos que combinando las sentencias de asignación, cualquier valor (cualquier entero no negativo) se puede asociar con una variable dada. Por ejemplo, la siguiente secuencia asigna el valor 3 a la variable `X` asignándole en primer lugar el valor cero y luego incrementando su valor tres veces:

```
clear X;
incr X;
incr X;
incr X;
```

Otra actividad muy común en los programas consiste en copiar datos de una ubicación a otra. En términos de Bare Bones, esto significa que tenemos que poder asignar el valor de una variable a otra variable. Esto se puede hacer poniendo a cero primero la variable de destino y luego incrementándola un número apropiado de veces. De hecho, ya hemos observado que la secuencia

Figura 12.4 Un programa en Bare Bones para el cálculo de $X \times Y$.

```
clear Z;
while X not 0 do;
   clear W;
   while Y not 0 do;
      incr Z;
      incr W;
      decr Y;
   end;
   while W not 0 do;
      incr Y;
      decr W;
   end;
   decr X;
end;
```

```
clear Z;
while X not 0 do;
  incr Z;
  decr X;
end;
```

transfiere a `Z` el valor asociado con `X`. Sin embargo, esta secuencia tiene el efecto colateral de destruir el valor original de `X`. Para corregir esto, podemos introducir una variable auxiliar a la que transferiremos primero el valor en cuestión desde su ubicación inicial. Luego, emplearemos esta variable auxiliar como origen de datos mediante el que restaurar la variable original, al mismo tiempo que colocamos el valor en cuestión en la variable de destino deseada. De esta manera, el movimiento de `Hoy` a `Mañana` puede realizarse mediante la secuencia mostrada en la Figura 12.5.

Adoptamos la sintaxis

```
copy nombre1 to nombre2;
```

(donde *nombre1* y *nombre2* representan nombres de variable) como notación abreviada para una estructura de sentencias de la forma mostrada en la Figura 12.5. Así, aunque el propio Bare Bones no tiene una sentencia de copia explícita, a menudo escribiremos programas como si la tuviera, asumiendo que para convertir esos programas informales en verdaderos programas Bare Bones es preciso sustituir las sentencias `copy` por sus estructuras equivalentes `while-end`, utilizando una variable auxiliar cuyo nombre no interfiera con ningún otro nombre utilizado en algún punto del programa.

La universalidad de Bare Bones

Apliquemos ahora la tesis de Church-Turing para confirmar nuestra afirmación de que Bare Bones es un lenguaje de programación universal. En primer lugar, observemos que cualquier programa escrito en Bare Bones puede considerarse como algo que está controlando el cálculo de una función. La entrada de la función está compuesta por los valores asignados a las variables antes de la ejecución del programa y la salida de la función está compuesta por los valores de las variables una vez que el programa termina. Para calcular la función, simplemente ejecutamos el programa comenzando con las asignaciones de variables adecuadas y luego observamos los valores de las variables una vez que el programa ha terminado.

Figura 12.5 Una implementación Bare Bones de la instrucción “copy Hoy to Mañana”.

```
clear Aux;
clear Mañana;
while Hoy not 0 do;
   incr Aux;
   decr Hoy;
end;
while Aux not 0 do;
   incr Hoy;
   incr Mañana;
   decr Aux;
end;
```

Bajo estas condiciones, el programa

```
incr X;
```

controla la computación de la misma función (la función sucesora) que hemos computado mediante el ejemplo de la máquina de Turing en la Sección 12.2. De hecho, incrementa el valor asociado con X en una unidad. De la misma forma, si interpretamos las variables X e Y como entradas y la variable Z como salida, el programa

```
copy Y to Z;
while X not 0 do;
 incr Z;
 decr X;
end;
```

controla el cálculo de la función suma.

Los investigadores han demostrado que el lenguaje de programación Bare Bones puede utilizarse para expresar algoritmos para calcular todas las funciones computables según Turing. Combinando esto con la tesis de Church-Turing, vemos que cualquier función computable podrá calcularse mediante un programa escrito en Bare Bones. Por tanto, Bare Bones es un lenguaje de programación universal en el sentido de que si existe un algoritmo para resolver un problema, entonces ese problema puede resolverse mediante algún programa escrito en Bare Bones. A su vez, eso implica que Bare Bones puede servir, desde el punto de vista teórico, como un lenguaje de programación de propósito general.

Decimos ***desde el punto de vista teórico*** porque un lenguaje así no sería desde luego tan cómodo como los lenguajes de alto nivel presentados en el Capítulo 6. Sin embargo, cada uno de esos lenguajes contiene, esencialmente, las características de Bare Bones como núcleo fundamental. De hecho, es ese núcleo fundamental lo que garantiza la universalidad de cada uno de esos lenguajes; todas las demás características de los distintos lenguajes se incluyen por comodidad.

Aunque no resultan prácticos en un entorno de programación de aplicaciones, los lenguajes como Bare Bones sí que tienen utilidad en las Ciencias teóricas de la computación. Por ejemplo, en el Apéndice E utilizaremos Bare Bones como herramienta para resolver la cuestión relativa a la equivalencia de las estructuras iterativas y recursivas, cuestión que hemos planteado en el Capítulo 5. Allí veremos que nuestra sospecha de que son equivalentes estaba, de hecho, justificada.

Cuestiones y ejercicios

1. Demuestre que la sentencia `invert X;` (cuya acción es convertir el valor de X a 0 si su valor inicial es distinto de cero y a 1 si su valor inicial es cero) se puede simular mediante un segmento de programa Bare Bones.
2. Demuestre que incluso nuestro lenguaje Bare Bones tan simple contiene más sentencias de las necesarias, demostrando que la sentencia `clear` puede sustituirse por combinaciones de otras sentencias del lenguaje.

3. Demuestre que la estructura `if-then-else` puede simularse utilizando Bare Bones. Es decir, escriba una secuencia de programa en Bare Bones que simule la acción de la sentencia

```
if X not 0 then S1 else S2;
```

donde S1 y S2 representan secuencias de sentencias arbitrarias.

4. Demuestre que cada una de las sentencias de Bare Bones puede expresarse en términos del lenguaje máquina del Apéndice C. (Por tanto, Bare Bones puede emplearse como lenguaje de programación para esa máquina.)
5. ¿Cómo podríamos tratar los números negativos en Bare Bones?
6. Describa la función computada mediante el siguiente programa Bare Bones, asumiendo que la entrada de la función está representada por X y su salida por Z:

```
clear Z;
while X not 0 do;
  incr Z;
  incr Z;
  decr X;
end;
```

12.4 Una función no computable

Ahora vamos a identificar una función que no es computable según Turing y por tanto, según la tesis de Church- Turing, tampoco será computable en sentido general. Por tanto, es una función cuyo cálculo cae más allá de las capacidades de las computadoras.

El problema de la detención

La función no computable de la que vamos a hablar está asociada con un problema conocido como **problema de la detención**, que (en un sentido informal) es el problema de tratar de predecir de antemano si un programa terminará (o se detendrá) si se inicia bajo ciertas condiciones. Por ejemplo, considere el siguiente programa simple en Bare Bones

```
while X not 0 do;
  incr X;
end;
```

Si ejecutamos este programa con un valor inicial de X igual a cero, el bucle no se ejecutará y la ejecución del programa terminará rápidamente. Sin embargo, si ejecutamos el programa con cualquier otro valor inicial de X, el bucle se ejecutará para siempre, por lo que obtendremos un proceso que no termina.

En este caso, entonces, es fácil concluir que la ejecución del programa se detendrá únicamente si se inicia asignando a X el valor cero. Sin embargo, cuando pasamos a estudiar problemas más complejos, la tarea de predecir el

comportamiento de un programa se vuelve más complicada. De hecho, en algunos casos, la tarea es imposible como veremos. Pero primero necesitamos formalizar nuestra terminología y centrar nuestro análisis de forma más precisa.

Nuestro ejemplo demuestra que el que un programa llegue a detenerse puede depender de los valores iniciales de sus variables. Por tanto, si esperamos poder predecir si la ejecución de un programa se detendrá, necesitamos ser muy precisos en relación a esos valores iniciales. La elección que estamos a punto de hacer con respecto a esos valores podrá parecerle extraña al lector a primera vista, pero no debe desesperarse. Nuestro objetivo es aprovecharnos de una técnica denominada **auto-referencia**, que es la idea de un objeto que se hace referencia a sí mismo. Esta estratagema ha conducido en muchas ocasiones a la obtención de resultados sorprendentes en matemáticas, que van desde curiosidades de tipo informal como la frase "Este enunciado es falso" a esa paradoja, mucho más seria, representada por la cuestión "¿El conjunto de todos los conjuntos se contiene a sí mismo?". Lo que estamos a punto de hacer, por tanto, es definir el escenario para una línea de razonamiento similar a "Si lo hace, entonces no lo hace; pero si no lo hace, entonces lo hace".

En nuestro caso, la auto-referencia se conseguirá asignando a las variables de un programa un valor inicial que representará al propio programa. Con este objetivo, observe que cada programa Bare Bones puede codificarse como un único patrón de bits de gran longitud en un formato en el que se utilice un carácter por byte aplicando el código ASCII, pudiendo después interpretarse ese patrón de bits como la representación binaria de un número entero no negativo (de magnitud bastante grande). Es ese valor entero el que asignaremos como valor inicial de las variables del programa.

Consideremos lo que sucedería si hiciéramos esto en el caso del programa sencillo

```
while X not 0 do;
  incr X;
end;
```

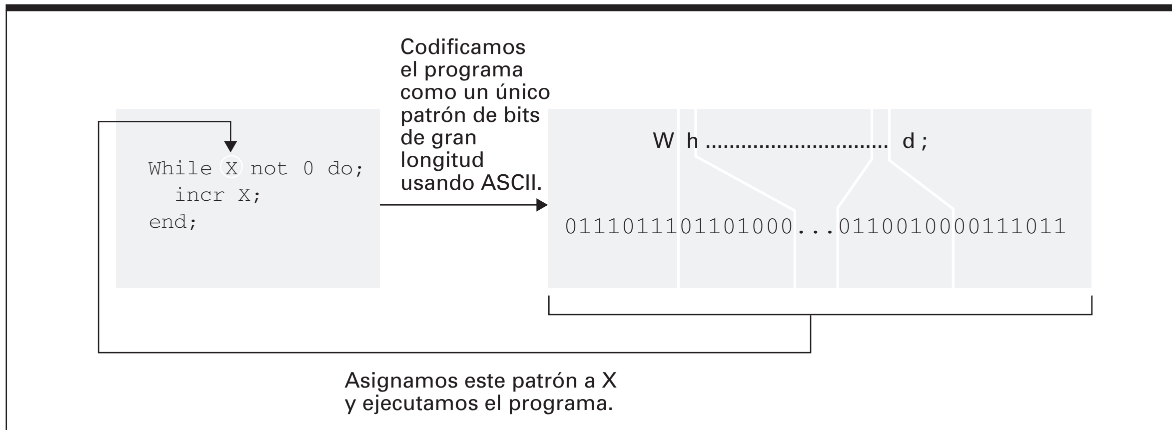
Deseamos saber lo que sucedería si iniciáramos este programa después de asignar a `X` el valor entero que represente al propio programa (Figura 12.6). En este caso, la respuesta es inmediata: puesto que `X` tiene un valor distinto de cero, el programa quedará atrapado en el bucle y nunca terminará. Por otro lado, si hiciéramos un experimento similar con el programa

```
clear X;
while X not 0 do;
  incr X;
end;
```

el programa terminaría, porque la variable `X` tendría el valor cero en el momento de alcanzar la estructura `while-end`, independientemente de cuál fuera su valor inicial.

Establezcamos entonces la siguiente definición: un programa Bare Bones es **auto-terminante** si al ejecutar el programa después de inicializar todas sus variables con la propia representación codificada del programa nos da un proceso que finaliza. Informalmente, un programa será auto-terminante si su eje-

Figure 12.6 Prueba de auto-terminación de un programa.



cución termina cuando se inicializa utilizando el propio programa como entrada. Esta es la auto-referencia que habíamos prometido.

Observe que el que un programa sea auto-terminante probablemente no tiene nada que ver con el propósito para el que fue escrito. Se trata simplemente de una propiedad que cada programa en Bare Bones poseerá o no poseerá. Es decir, cada programa Bare Bones será auto-terminante o no lo será.

Ahora podemos describir el problema de la detención de una forma precisa: será el problema de determinar si los programas Bare Bones son o no auto-terminantes. En breve veremos que no existe ningún algoritmo para responder a esta cuestión de carácter general. Es decir, no existe ningún algoritmo que, al darle como entrada cualquier programa Bare Bones sea capaz de determinar si ese programa es o no auto-terminante. Por tanto, la solución al problema de la detención cae más allá de las capacidades de la computadora.

El hecho de que hayamos resuelto aparentemente el problema de la detención en los ejemplos anteriores y ahora digamos que el problema de la detención es irresoluble puede sonar contradictorio, así que hagamos una pausa para clarificar los conceptos. Las observaciones que hemos utilizado en los ejemplos eran aplicables a esos casos particulares y no serían aplicables a todas las situaciones. Lo que el problema de la detención exige es un único algoritmo genérico que pueda aplicarse a cualquier programa Bare Bones para determinar si es auto-terminante. Nuestra capacidad de aplicar ciertos conocimientos aislados para determinar si un programa concreto es auto-terminante, no implica de ninguna forma la existencia de una única solución genérica que pueda aplicarse en todos los casos. En resumen, podemos construir una máquina que pueda resolver el problema de la detención en un caso concreto, pero no podemos construir una única máquina que pudiéramos emplear para resolver el problema de la detención en cualquier situación que surja.

La irresolubilidad del problema de la detención

Ahora queremos demostrar que resolver el problema de la detención cae más allá de las capacidades de las máquinas. Nuestro enfoque consistirá en demostrar que el resolver el problema requeriría un algoritmo para calcular una fun-

ción no computable. Las entradas de la función en cuestión son versiones codificadas de programas Bare Bones; sus salidas están limitadas a los valores 0 y 1. Para ser más precisos, definiremos la función de modo que una entrada que represente un programa auto-terminante genere el valor de salida 1, mientras que una entrada que represente un programa que no sea auto-terminante genere el valor de salida 0. En aras de la concesión, nos referiremos a esta función como ***función de detención.***

Nuestra tarea consiste en demostrar que la función de detención no es computable. El enfoque que adoptaremos es una técnica conocida con el nombre de "prueba por contradicción". En resumen, demostraremos que un enunciado es falso demostrando que no puede ser verdadero. Demostramos, por tanto, que el enunciado "La función de detención es computable" no puede ser cierto. Nuestro argumento completo se resume en la Figura 12.7.

Si la función de detención es computable, entonces (como Bare Bones es un lenguaje de programación universal) debe existir un programa Bare Bones que la calcule. En otras palabras, existe un programa Bare Bones que terminará con su salida igual a 1 si su entrada es la versión codificada de un programa auto-terminante, y en caso contrario terminará con su salida igual a 0.

Para aplicar este programa no necesitamos identificar qué variable utilizamos como entrada, sino que basta simplemente con inicializar todas las variables del programa con la representación codificada del programa que hay que probar. Esto se debe a que una variable que no sea una variable de entrada será, inherentemente, una variable cuyo valor inicial no afecta al valor final de salida. Podemos entonces concluir que si la función de detención es computable, entonces existirá un programa Bare Bones que termine con su salida igual a 1 si se inicializan todas las variables con la versión codificada de un programa auto-terminante y que termine con su salida igual a 0 en caso contrario.

Asumiendo que la variable de salida del programa es `X` (si no fuera así podemos simplemente renombrar las variables), podríamos modificar el programa añadiendo las sentencias

```
while X not 0 do;
end;
```

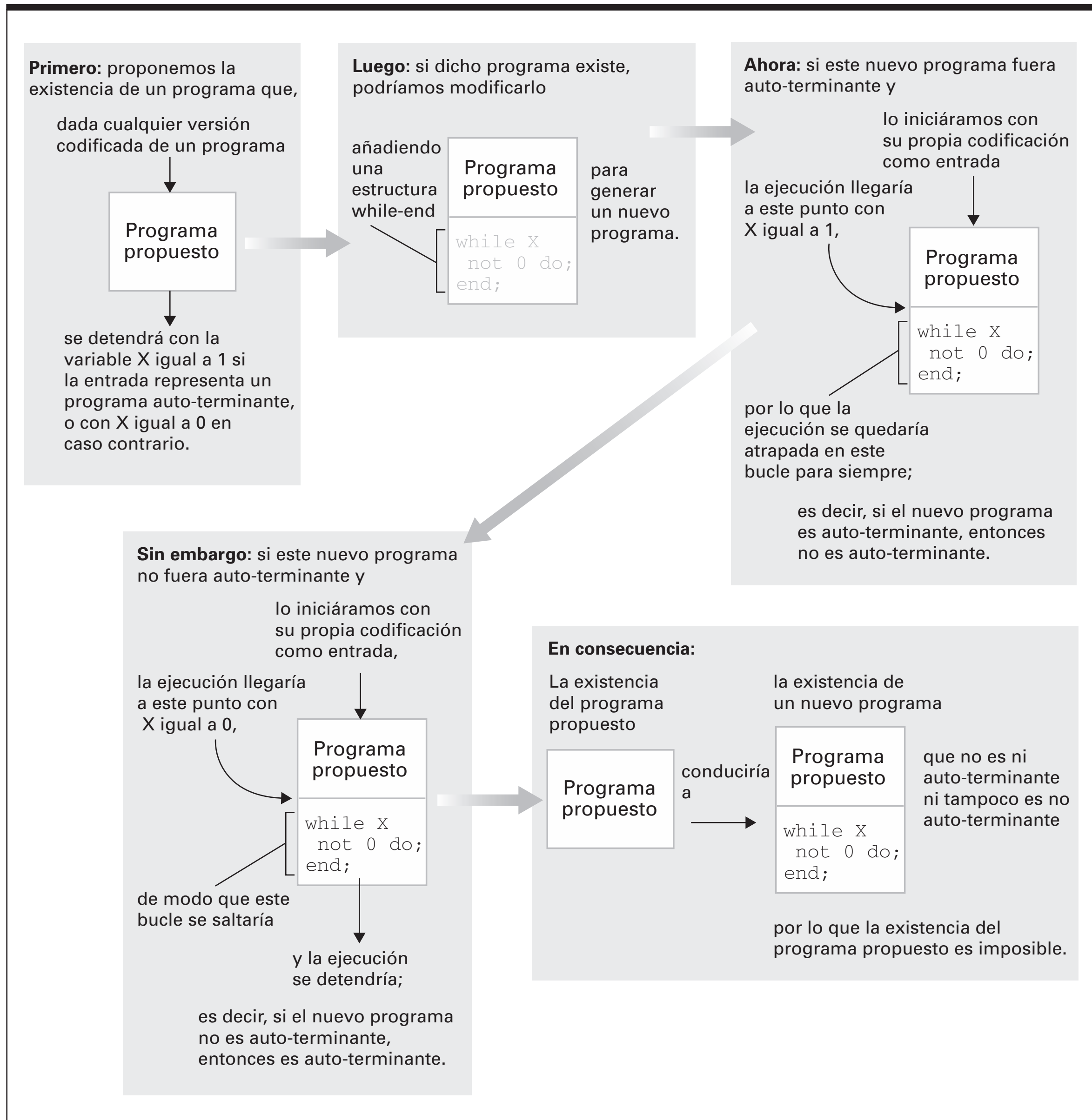
al final, obteniendo así un nuevo programa. Este nuevo programa deberá ser auto-terminante o no. Sin embargo, estamos a punto de ver que no puede ser ninguna de las dos cosas.

En particular, si este nuevo programa fuera auto-terminante y lo ejecutáramos después de inicializar sus variables con la propia representación codificada del programa, entonces cuando su ejecución alcanzara la sentencia `while` que hemos añadido, la variable `X` contendría un 1. (Hasta este punto el nuevo programa es idéntico al programa original que generaba un 1 si su entrada era la representación de un programa auto-terminante.) Llegados a este punto, la ejecución del programa quedaría atrapada para siempre en la estructura `while-end`, porque no hemos incluido ninguna sentencia para decrementar `X` dentro del bucle. Pero esto contradice nuestra suposición de que el nuevo programa es auto-terminante. Por tanto, tenemos que concluir que el nuevo programa no es auto-terminante.

Sin embargo, si el nuevo programa fuera no auto-terminante y lo ejecutáramos después de inicializar sus variables con la representación codificada del

propio programa, llegaría a la sentencia `while` que hemos añadido con un valor 0 asignado a la variable `X`. (Esto ocurre porque las sentencias que preceden a la sentencia `while` constituyen el programa original, que genera una salida igual a `0` cuando su entrada representa un programa que no es auto-terminante.) En este caso, nos saltaríamos el bucle de la estructura `while-end` y el programa se detendría. Pero esta es precisamente la propiedad que caracteriza a un programa auto-terminante, así que estamos obligados a concluir que el nuevo programa es auto-terminante, al igual que nos vimos forzados anteriormente a concluir que el programa no es auto-terminante.

Figure 12.7 Demostración de la irresolubilidad del problema de la detención.



En resumen, vemos que nos encontramos en la situación imposible de un programa que por un lado debe ser auto-terminante o no serlo, y que por otro lado no puede ser ninguna de las dos cosas. En consecuencia, la suposición que nos ha conducido a este dilema tiene que ser falsa.

Podemos concluir que la función de detención no es computable, y como la solución al problema de la detención depende de la computación de dicha función, llegamos a la conclusión de que resolver el problema de la detención cae más allá de las capacidades de cualquier sistema algorítmico. Este tipo de problemas se denominan **problemas irresolubles**.

Para terminar, conviene relacionar lo que acabamos de ver con las ideas del Capítulo 11. Allí, una de las principales cuestiones subyacentes era si la potencia de las máquinas de computación incluía las capacidades requeridas para la propia inteligencia. Recuerde que las máquinas solo pueden resolver problemas que tengan solución algorítmica, y ahora acabamos de ver que existen problemas que no tienen solución algorítmica. La cuestión es, entonces, si la mente humana abarca algo más que la ejecución de procesos algorítmicos. Si no es así, entonces los límites que acabamos de identificar aquí serán también los límites del pensamiento humano. No hace falta decir que esta cuestión es altamente debatida y en ocasiones ese debate llega a ser bastante emocional. Por ejemplo, si la mente humana no es más que una máquina programada, entonces nos veríamos forzados a concluir que los seres humanos no poseen libre albedrío.

Cuestiones y ejercicios

1. ¿Es auto-terminante el siguiente programa Bare Bones? Explique su respuesta.

```
incr X;
decr Y;
```

2. ¿Es auto-terminante el siguiente programa Bare Bones? Explique su respuesta.

```
copy X to Y;
incr Y;
incr Y;
while X not 0 do;
  decr X;
  decr X;
  decr Y;
  decr Y;
end;
decr Y;
while Y not 0 do;
end;
```

3. ¿Qué error hay en el siguiente escenario?

En un cierto pueblo, todo el mundo es propietario de su casa. El pintor del pueblo afirma que él pinta todas aquellas casas que no son pintadas por sus propios dueños y que solo pinta las casas que no son pintadas por sus propios dueños.

(*Pista:* ¿quién pinta la casa del pintor?)

12.5 Complejidad de los problemas

En la Sección 12.4 hemos investigado la resolubilidad de los problemas. En esta sección, estamos interesados en la cuestión de si un problema resoluble tiene una solución práctica. Veremos que algunos problemas que son teóricamente resolubles son tan complejos que los podemos considerar como irresolubles desde el punto de vista práctico.

Medida de la complejidad de un problema

Comenzaremos volviendo a nuestro estudio de la eficiencia de los algoritmos que iniciamos en la Sección 5.6. Allí utilizábamos la notación zeta-mayúscula para clasificar los algoritmos de acuerdo con el tiempo requerido para ejecutarlos. Vimos allí que el algoritmo de ordenación por inserción pertenece a la clase $\Theta(N^2)$, el algoritmo de búsqueda secuencial pertenece a la clase $\Theta(n)$ y que el algoritmo de búsqueda binaria es de la clase $\Theta(\lg n)$. Utilizaremos ahora este sistema de clasificación como ayuda para identificar la complejidad de los problemas. Nuestro objetivo es desarrollar un sistema de clasificación que nos diga qué problemas son más complejos que otros y, en último término, qué problemas son tan complejos que su solución es imposible desde el punto de vista práctico.

La razón de que nuestro presente análisis se base en nuestro conocimiento de la eficiencia de los algoritmos es que queremos medir la complejidad de un problema en términos de la complejidad de su solución. Consideraremos que un problema simple es aquel que dispone de una solución simple y que un problema complejo es aquel que no dispone de una solución simple. Observe que el hecho de que un problema tenga una solución difícil no implica necesariamente que el propio problema sea complejo. Después de todo, un problema dispone de múltiples soluciones, y alguna de ellas tenderá a ser compleja. Por tanto, para concluir que un problema es complejo tenemos que demostrar que ninguna de sus soluciones es simple.

En las Ciencias de la computación, los problemas que nos interesan son aquellos que son resolubles mediante máquinas. Las soluciones a estos problemas se expresan en forma de algoritmos. Por tanto, la complejidad de un problema estará determinada por las propiedades de los algoritmos que permiten resolver dicho problema. Para ser más precisos, consideraremos que la complejidad de un problema es igual a la complejidad del algoritmo más simple que permite resolverlo.

Pero, ¿cómo medimos la complejidad de un algoritmo? Lamentablemente, el término ***complejidad*** tiene diferentes interpretaciones. Una de esas inter-

pretaciones trata con la cantidad de toma de decisiones y el grado de ramificación que contenga el algoritmo. Desde ese punto de vista, un algoritmo complejo sería aquel que incluyera un conjunto de instrucciones muy entrelazado y complicado. Esta interpretación puede ser compatible con el punto de vista de un ingeniero de software, que está interesado en los problemas relativos al descubrimiento y representación de algoritmos, pero no captura el concepto de complejidad desde el punto de vista de una máquina. Una máquina no toma decisiones realmente a la hora de seleccionar la siguiente instrucción que tiene que ejecutar, sino que simplemente sigue su ciclo de máquina una y otra vez, ejecutando cada vez la instrucción que le indique el contador de programa. En consecuencia, una máquina puede ejecutar un conjunto de instrucciones muy entrelazado con la misma facilidad que puede ejecutar una lista de instrucciones en un simple orden secuencial. Esta interpretación de la complejidad tiende, por tanto, a medir la dificultad inherente a la representación de un algoritmo, más que la complejidad del propio algoritmo.

Una interpretación que refleja con más precisión la complejidad de un algoritmo desde el punto de vista de una máquina consiste en medir el número de pasos que hay que realizar a la hora de ejecutar el algoritmo. Observe que esto no es lo mismo que el número de instrucciones que aparecen en el programa escrito. Un bucle cuyo cuerpo esté compuesto de una única instrucción, pero cuyo control solicite la ejecución del bucle 100 veces es equivalente a 100 instrucciones secuenciales desde el punto de vista del número de instrucciones ejecutadas. Por tanto, esa rutina del bucle se consideraría más compleja que una lista de 50 instrucciones escritas individualmente, aún cuando esta lista parezca más larga en forma escrita. Lo importante aquí es que este significado de la palabra ***complejidad*** está relacionado en último término con el tiempo que una máquina tarda en ejecutar una solución y no con el tamaño del programa que representa dicha solución.

Podemos por tanto considerar que un problema es complejo si todas sus soluciones requieren una gran cantidad de tiempo de ejecución. Esta definición de la complejidad se suele denominar **complejidad temporal**. Ya nos hemos topado indirectamente con el concepto de complejidad temporal al estudiar el tema de la complejidad de los algoritmos en la Sección 5.6. Después de todo, el estudio de la eficiencia de un algoritmo es el estudio de la complejidad temporal de ese algoritmo, los dos conceptos son simplemente inversos el uno del otro. Es decir, "más eficiente" es lo mismo que "menos complejo". Por tanto, en términos de complejidad temporal, el algoritmo de búsqueda secuencial (que vimos que es del orden $\Theta(n)$) es una solución más compleja del problema de buscar en una lista que el algoritmo de búsqueda binaria (que vimos que es del orden de $\Theta(\lg n)$).

Apliquemos ahora nuestros conocimientos sobre la complejidad de los algoritmos para obtener un medio de identificar la complejidad de los problemas. Definimos la complejidad (temporal) de un problema como $\Theta(f(n))$, donde $f(n)$ es alguna expresión matemática dependiente de n, si existe un algoritmo para resolver el problema cuya complejidad temporal sea del orden $\Theta(f(n))$ y ningún otro algoritmo para resolver el problema tiene una complejidad temporal menor. Es decir, la complejidad (temporal) de un problema se define como la complejidad (temporal) de su mejor solución. Lamentablemente, encontrar la mejor solución a un problema y saber que es la mejor solu-

ción suele ser a menudo bastante complicado. En dichas situaciones, se utiliza una variante de la notación zeta-mayúscula, denominada **notación O mayúscula**, para representar lo que se conoce acerca de la complejidad de un problema. De forma más precisa, si $f(n)$ es una expresión matemática dependiente de n y si un problema puede ser resuelto por un algoritmo en $\Theta(f(n))$, diremos entonces que el problema es del orden $O(f(n))$. Por tanto, decir que un problema pertenece a la clase $O(f(n))$ implica que tiene una solución cuya complejidad es del orden $\Theta(f(n))$ pero que posiblemente podría tener una solución mejor.

Nuestro análisis de los algoritmos de búsqueda y ordenación nos dice que el problema de buscar en una lista de longitud n (cuando lo único que sabemos es que la lista ha sido previamente ordenada) es del orden $O(\lg n)$, ya que el algoritmo de búsqueda binaria permite resolver el problema. Además, los investigadores han demostrado que el problema de búsqueda es en realidad de orden $\Theta(\lg n)$, por lo que la búsqueda binaria representa una solución óptima para dicho problema. Por el contrario, sabemos que el problema de ordenar una lista de longitud n (cuando no sabemos nada acerca de la distribución original de los valores que la componen) es del orden $O(n^2)$, porque el algoritmo de ordenación por inserción permite resolver ese problema. Sin embargo, se sabe que el problema de la ordenación es de orden $\Theta(n \lg n)$, lo que nos dice que el algoritmo de ordenación por inserción no es una solución óptima (en términos de la complejidad temporal).

Un ejemplo de una mejor solución al problema de ordenación es el algoritmo de ordenación por combinación. El enfoque de ese algoritmo consiste en ordenar pequeñas partes de la lista con el fin de obtener partes ordenadas más grandes, que luego pueden combinarse para obtener otras partes ordenadas aún mayores. Cada proceso de combinación aplica el algoritmo de combinación que ya presentamos al hablar de los archivos secuenciales (Figura 9.15). Por comodidad, lo volvemos a mostrar en la Figura 12.8, esta vez en el contexto de la combinación de dos listas. El algoritmo completo (recursivo) de ordenación

Figura 12.8 Un procedimiento CombinarListas para combinar dos listas.

```
procedure CombinarListas (ListaEntradaA, ListaEntradaB, ListaSalida)

if (ambas listas de entrada están vacías) then (Parar con ListaSalida vacía)
if (ListaEntradaA está vacía)
   then (Declarar que la lista se ha terminado)
   else (Declarar como entrada actual su primera entrada)
if (ListaEntradaB está vacía)
   then (Declarar que la lista se ha terminado)
   else (Declarar como entrada actual su primera entrada)
while (ninguna de las listas de entrada se ha terminado) do
      (Poner la entrada actual "más pequeña" en ListaSalida;
       if (esa entrada actual es la última entrada en su correspondiente lista de entrada)
           then (Declarar que la lista de entrada se ha terminado)
           else (Declarar la siguiente entrada de esa lista de entrada como la entrada actual de la lista)
      )

Comenzando con la entrada actual de la lista de entrada que no se haya terminado,
      copiar las entradas restantes a ListaSalida.
```

por combinación se incluye como un procedimiento denominado `OrdenacionComb` en la Figura 12.9. Al pedirle que ordene una lista, este procedimiento comprueba primero si la lista tiene menos de dos entradas. En caso afirmativo, la tarea del procedimiento estará completada. En caso contrario, el procedimiento divide la lista en dos fragmentos, pide a otras copias del procedimiento `OrdenacionComb` que ordene esos fragmentos y luego combina esos fragmentos ordenados para obtener la versión final ordenada de la lista.

Para analizar la complejidad de este algoritmo, consideraremos en primer lugar el número de comparaciones que hay que realizar entre las entradas de las listas a la hora de combinar una lista de longitud r con una lista de longitud s. El proceso de combinación se lleva a cabo comparando repetidamente una entrada de una lista con una entrada de la otra lista y colocando la "más pequeña" de las dos entradas en la lista de salida. Por tanto, cada vez que se efectúa una comparación, el número de entradas que queda por considerar se reduce en una unidad. Puesto que solo hay $r + s$ entradas al principio, podemos concluir que el proceso de combinar dos listas requerirá no más de $r + s$ comparaciones.

Consideremos ahora el algoritmo completo de ordenación por combinación. Este algoritmo afronta la tarea de ordenar una lista de longitud n de tal forma que el problema de ordenación inicial se reduce a otros problemas más pequeños, en cada uno de los cuales tenemos que ordenar una lista de longitud aproximadamente igual a $n/2$. Estos dos problemas se reducen a su vez a un total de cuatro problemas de ordenación de una lista de longitud aproximadamente igual a $n/4$. Este proceso de división puede resumirse mediante la estructura de árbol de la Figura 12.10, donde cada nodo del árbol representa un único problema dentro del proceso recursivo y las ramas situadas por debajo de un nodo representan los problemas más pequeños derivados del problema padre. Por tanto, podemos averiguar el número total de comparaciones que se realizan a lo largo de todo el proceso de ordenación sumando el número de comparaciones que tienen lugar en los distintos nodos del árbol.

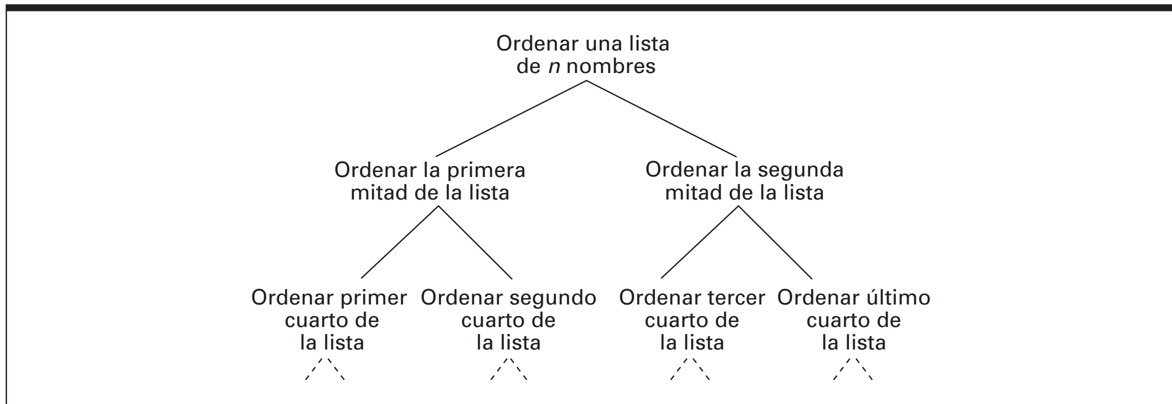
Determinemos primero el número de comparaciones realizadas en cada nivel del árbol. Observe que cada uno de los nodos que aparece en un nivel determinado del árbol está encargado de ordenar una parte diferente de la lista general. Esa ordenación se lleva a cabo mediante el proceso de combinación y requiere, por tanto, un número de comparaciones que no es superior al de entradas existentes en el segmento de lista, como ya hemos demostrado. Por tanto, cada nivel del árbol requiere un número de comparaciones que no es mayor que el número total de entradas de todos los segmentos de lista y, como

Figura 12.9 El algoritmo de ordenación por combinación implementado como un procedimiento OrdenacionComb.

```
procedure OrdenacionComb (Lista)

if (Lista tiene más de una entrada)
  then (Aplicar el procedimiento OrdenacionComb para ordenar la primera mitad de Lista;
        Aplicar el procedimiento OrdenacionComb para ordenar la segunda mitad de Lista;
        Aplicar el procedimiento CombinarListas para combinar la primera y segunda
            mitades de Lista, para generar una versión ordenada de Lista
       )
```

Figura 12.10 La jerarquía de problemas generada por el algoritmo de ordenación por combinación.



los segmentos de un determinado nivel del árbol representan partes disjuntas de la lista original, ese total no será mayor que la longitud de la lista original. En consecuencia, cada nivel del árbol requiere no más de n comparaciones. (Por supuesto, el nivel inferior implica ordenar la lista de longitud menor que dos, lo que no requiere ninguna comparación en absoluto.)

Ahora vamos a calcular el número de niveles del árbol. Para ello, observe que el proceso de dividir los problemas en otros problemas más pequeños continúa hasta obtener listas de longitud inferior a dos. Por tanto, el número de niveles del árbol estará determinado por el número de veces que, comenzando con el valor inicial n, podamos dividir entre dos repetidamente hasta que el resultado no sea mayor que uno, lo que nos da un resultado de lg n. De forma más precisa, no hay más de $\lceil \lg n \rceil$ niveles del árbol que requieren comparaciones, donde la notación $\lceil \lg n \rceil$ representa el valor de lg n redondeado al siguiente entero superior.

Finalmente, el número total de comparaciones realizadas por el algoritmo de ordenación por combinación a la hora de ordenar una lista de longitud n se obtiene multiplicando el número de comparaciones realizadas en cada nivel del árbol por el número de niveles en el que se realizan comparaciones. Podemos concluir que ese valor no es superior a $n\lceil \lg n \rceil$. Puesto que el grafo de $n\lceil \lg n \rceil$ tiene la misma forma general que el grafo de $n \lg n$, podemos concluir que el algoritmo de ordenación por combinación pertenece a $O(n \lg n)$. Combinando esto con el hecho de que los investigadores nos dicen que el problema de ordenación tiene una complejidad igual a $\Theta(n \lg n)$, querrá decir que el algoritmo de ordenación por combinación representa una solución óptima al problema de ordenación.

Problemas polinómicos y no polinómicos

Suponga que $f(n)$ y $g(n)$ son expresiones matemáticas. Decir que $g(n)$ está acotada por $f(n)$ significa que, a medida que aplicamos esas expresiones a valores cada vez mayores de n, el valor de $f(n)$ termina por ser mayor que el de $g(n)$ y continúa siendo mayor que $g(n)$ para todos los valores n superiores. En otras palabras, decir que $g(n)$ está acotada por $f(n)$ significa que la gráfica de $f(n)$

Complejidad espacial

Una alternativa a medir la complejidad en términos del tiempo es la de medir en su lugar los requisitos de espacio de almacenamiento, lo que da como resultado una medida que se conoce con el nombre de **complejidad espacial**. Es decir, la complejidad espacial de un problema está determinada por la cantidad de espacio de almacenamiento requerida para solucionar el problema. En el texto hemos visto que la complejidad temporal de ordenar una lista con n entradas es $O(n \lg n)$. La complejidad espacial del mismo problema no es superior a $O(n + 1) = O(n)$. Después de todo, ordenar una lista con n entradas utilizando la ordenación por inserción requiere reservar espacio para la propia lista más el espacio necesario para almacenar una única entrada de una forma temporal. Por tanto, si nos pidieran almacenar listas cada vez más largas nos encontraríamos con que el tiempo requerido se iría incrementando más rápidamente que el espacio necesario. Este es, de hecho, un fenómeno bastante común. Puesto que el utilizar el espacio requiere un cierto tiempo, la complejidad espacial de un problema nunca crece más rápidamente que su complejidad temporal.

A menudo se pueden adoptar compromisos entre la complejidad temporal y la espacial. En algunas aplicaciones, resulta ventajoso realizar ciertos cálculos de antemano y almacenar los resultados en una tabla, de la que se los pueda extraer rápidamente cuando sea necesario. Esa técnica de "búsqueda en tablas" reduce el tiempo requerido para obtener un resultado cuando se necesita, a expensas del espacio adicional ocupado por la tabla. Por otro lado, a menudo se emplea tambien la compresión de datos para reducir los requisitos de almacenamiento, a expensas del tiempo adicional requerido para comprimir y descomprimir los datos.

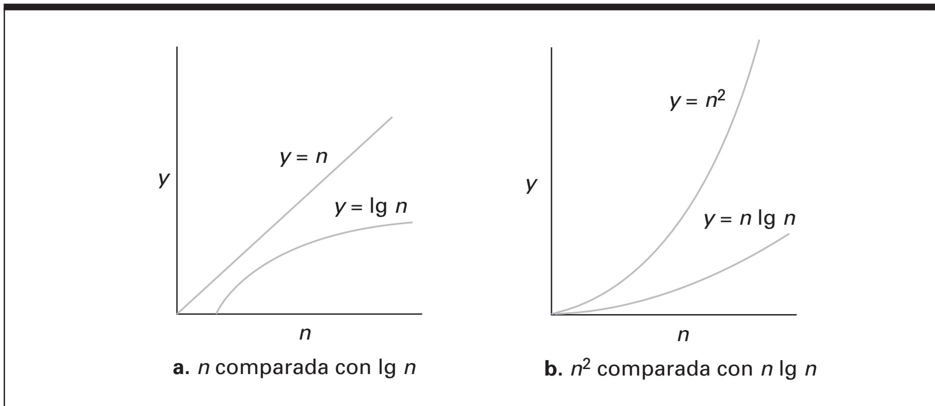
estará situada por encima de la gráfica de $g(n)$ para valores "grandes" de n. Por ejemplo, la expresión $\lg n$ está acotada por la expresión n (Figura 12.11a) y $n \lg n$ está acotada por n^2 (Figura 12.11b).

Decimos que un problema es un **problema polinómico** si pertenece a la clase $O(f(n))$, en la que la expresión $f(n)$ es un polinomio ella misma o está acotada por un polinomio. El conjunto de todos los problemas polinómicos se denomina **P**. Observe que nuestros anteriores análisis nos dicen que los problemas de buscar en una lista y de ordenar una lista pertenecen a P.

Decir que un problema es un problema polinómico es un enunciado del tiempo necesario para resolver el problema. A menudo, decimos que un problema perteneciente a P puede resolverse en tiempo polinómico o que el problema tiene una solución de tiempo polinómico.

Identificar los problemas pertenecientes a P tiene una importancia crucial en las Ciencias de la computación, porque esa cuestión está estrechamente relacionada con otra serie de cuestiones relativas a si los problemas tienen soluciones prácticas. De hecho, los problemas que no pertenecen a la clase P se caracterizan por tener tiempos de ejecución extremadamente largos, incluso para entradas de tamaño moderado. Por ejemplo, considere un problema cuya solución requiera 2^n pasos. La expresión exponencial 2^n no está acotada por ningún polinomio, si $f(n)$ es un polinomio, entonces al ir incrementando el valor de n, nos encontramos con que los valores de 2^n terminarán por ser mayores que los de $f(n)$. Esto significa que un algoritmo de complejidad $\Theta(2^n)$ será generalmente menos eficiente (y requerirá por tanto más tiempo) que un

Figura 12.11 Gráficas de las expresiones matemáticas *n*, lg *n*, *n* lg *n* y n^2.



algoritmo de complejidad $\Theta(f(n))$. Cuando la complejidad de un algoritmo se identifica mediante una expresión exponencial, se dice que el algoritmo requiere un tiempo exponencial.

Como ejemplo concreto, considere el problema de enumerar todos los posibles subcomités que pueden formarse a partir de un grupo formado por n personas. Puesto que hay $2^n - 1$ subcomités posibles (permitimos que un subcomité esté compuesto por el conjunto completo, pero no consideramos el subconjunto vacío como un subcomité), cualquier algoritmo que resuelva este problema deberá tener al menos $2^n - 1$ pasos y, por tanto, su complejidad será por lo menos de ese orden. Pero, la expresión $2^n - 1$, al ser exponencial, no está acotada por ningún polinomio. Por tanto, cualquier solución a este problema es enormemente costosa en términos de tiempo a medida que se incrementa el tamaño del grupo a partir del cual se forman los comités.

A diferencia del problema de los subcomités, cuya complejidad es grande simplemente debido al tamaño de su salida, existen problemas cuya complejidad es grande aún cuando la salida final sea simplemente una respuesta de tipo sí o no. Un ejemplo sería la capacidad de responder cuestiones acerca de la verdad o falsedad de enunciados relativos a la suma de números reales. Por ejemplo, podemos reconocer fácilmente que la respuesta a la cuestión "¿Es verdad que existe un número real, que al sumarse consigo mismo da el valor 6?" es sí, mientras que la respuesta a la pregunta "¿Es verdad que existe un número real distinto de cero que, al sumarse consigo mismo da 0?" es no. Sin embargo, a medida que esas cuestiones se vuelven más complejas, nuestra capacidad de responderlas comienza a desvanecerse. Si nos tuviéramos que enfrentar con muchas de esas preguntas nos podríamos sentir tentados a recurrir a un programa de computadora en busca de ayuda. Lamentablemente, se ha demostrado que la capacidad de responder a estas cuestiones requiere un tiempo exponencial, por lo que incluso una computadora terminará por no poder dar una respuesta rápida, a medida que las cuestiones vayan haciéndose más complejas.

El hecho de que los problemas teóricamente resolubles pero que no pertenecen a P tengan una complejidad temporal tan enorme nos lleva a concluir que esos problemas son esencialmente irresolubles desde un punto de vista

práctico. Los expertos en Ciencias de la computación denominan a estos problemas **intratables**. A su vez, la clase P ha llegado a representar una importante frontera que diferencia a los problemas intratables de aquellos que pueden tener soluciones prácticas. Por tanto, la comprensión de la clase P ha llegado a ser un objetivo de gran importancia dentro de las Ciencias de la computación.

Problemas NP

Consideremos ahora el **problema del viajante** relativo a un viajante que debe visitar a cada uno de sus clientes que viven en ciudades distintas, sin superar el presupuesto que tiene asignado para el viaje. Su problema entonces es encontrar una ruta (que comience en su ciudad de residencia, que conecte todas las ciudades necesarias y que vuelva a su ciudad de residencia) cuya longitud total no exceda del número de máximo de kilómetros que puede viajar.

La solución tradicional a este problema es considerar todas las rutas potenciales de manera sistemática, comparando la longitud de cada ruta con el límite de kilómetros hasta encontrar una ruta aceptable o hasta haber analizado todas las posibilidades. Sin embargo, este enfoque no permite obtener una solución de tiempo polinómico. A medida que se incrementa el número de ciudades, la cantidad de rutas que hay que probar crece más rápidamente que cualquier polinomio. Por tanto, resolver el problema del viajante de esta forma no es práctico en aquellos casos en los que el número de ciudades implicadas sea grande.

En conclusión, para resolver el problema del viajante en una cantidad de tiempo razonable necesitamos encontrar un algoritmo más rápido. Nuestro deseo se ve alimentado por la observación de que si existe una ruta satisfactoria y por causalidad la seleccionamos en primer lugar, nuestro algoritmo actual terminará rápidamente. En particular, la siguiente lista de instrucciones puede ejecutarse rápidamente y tendría el potencial de resolver el problema:

```
Seleccionar una de las rutas posibles y calcular su
distancia total.
If (esta distancia no es mayor que el número máximo de
        kilómetros)
    then (declarar que la búsqueda ha tenido éxito)
    else (no declarar nada)
```

Sin embargo, este conjunto de instrucciones no es un algoritmo en el sentido técnico de la palabra. Su primera instrucción es ambigua, ya que no especifica qué ruta hay seleccionar y tampoco especifica cómo hay que tomar la decisión. En lugar de ello, la instrucción depende de la creatividad que tenga el mecanismo que esté ejecutando el programa a la hora de tomar la decisión por sí mismo. A este tipo de instrucciones las llamamos no deterministas y a un "algoritmo" que contenga tal clase de instrucciones le denominamos **algoritmo no determinista**.

Observe que cuando el número de ciudades aumenta, el tiempo requerido para ejecutar el algoritmo no determinista anterior crece de forma relativamente lenta. El proceso de seleccionar una ruta consiste simplemente en gene-

rar una lista de las ciudades, lo cual puede llevarse a cabo en un tiempo proporcional al número de ciudades existente. Además, el tiempo requerido para calcular la distancia total a lo largo de la ruta seleccionada también es proporcional al número de ciudades que hay que visitar, y el tiempo requerido para comparar este total con el número máximo de kilómetros es independiente del número máximo de ciudades. Por ello, el tiempo requerido para ejecutar ese algoritmo no determinista estará acotado por un polinomio. Por tanto, es posible resolver el problema del viajante en un tiempo polinómico utilizando un algoritmo no determinista.

Por supuesto, esta solución no determinista no es una solución totalmente satisfactoria. Depende de que acertemos a la hora de elegir la ruta. Pero su existencia es suficiente para sugerir que quizá exista una solución determinista al problema del viajante capaz de ejecutarse en tiempo polinómico. El que esto sea o no cierto sigue siendo una cuestión aún no resuelta. De hecho, el problema del viajante es uno de entre muchos problemas que se sabe que dispone de soluciones no deterministas capaces de ejecutarse en tiempo polinómico, pero para los cuales no se conoce aún ninguna solución determinista que se ejecute en tiempo polinómico. La llamativa eficiencia de las soluciones no deterministas a estos problemas hace que algunos expertos tengan la esperanza de poder encontrar algún día soluciones deterministas eficientes, aunque la mayoría creen que estos problemas son simplemente demasiado complejos como para resolverlos utilizando algoritmos deterministas eficientes.

Un problema que pueda resolverse en tiempo polinómico mediante un algoritmo no determinista se denomina **problema polinómico no determinista** o **problema NP** (*Nondeterministic Polynomial*). Habitualmente se denomina **NP** a la clase formada por todos los problemas de tipo NP. Observe que todos los problemas pertenecientes a P también pertenecen a NP, porque podemos añadir una instrucción no determinista a cualquier algoritmo (determinista) sin que el rendimiento de este se vea afectado.

El que todos los problemas NP pertenezcan también a P sigue siendo una cuestión no resuelta como demuestra el problema del viajante. Este es quizá el problema actualmente no resuelto más conocido de todo el campo de las Ciencias de la computación. Su solución podría tener consecuencias muy importantes. Por ejemplo, en la siguiente sección veremos que se han diseñado sistemas de cifrado cuya integridad depende de la enorme cantidad de tiempo requerido para resolver problemas similares al problema del viajante. Si resultara que existen soluciones eficientes a dicho problema, esos sistemas de cifrado se verían comprometidos.

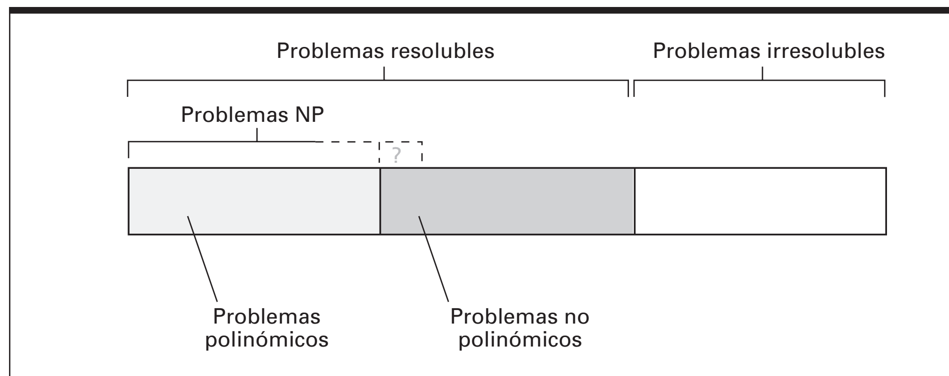
Los esfuerzos para resolver la cuestión de si la clase NP coincide, de hecho, con la clase P ha conducido al descubrimiento de una clase de problemas dentro de la clase NP a los que se conoce con el nombre de **problemas NP-completos**. Estos problemas tienen la propiedad de que una solución en tiempo polinómico para cualquiera de ellos proporcionaría también una solución en tiempo polinómico para todos los demás problemas de la clase NP. Es decir, si se pudiera encontrar un algoritmo (determinista) que resolviera uno de los problemas NP-completos en tiempo polinómico, entonces dicho algoritmo podría ampliarse para resolver cualquier otro problema de la clase NP en tiempo polinómico. A su vez, eso implicaría que la clase NP coincide con la clase P. El problema del viajante es un ejemplo de problema NP-completo.

En resumen, hemos visto que los problemas pueden clasificarse como resolubles (que tienen una solución algorítmica) o irresolubles (que no tienen una solución algorítmica), como se ilustra en la Figura 12.12. Además, dentro de la clase de problemas resolubles existen dos subclases. Una de ellas es el conjunto de problemas polinómicos, que contiene aquellos problemas que disponen de soluciones prácticas. El segundo es el conjunto de los problemas no polinómicos, cuyas soluciones solo son prácticas para entradas relativamente pequeñas o para entradas cuidadosamente seleccionadas. Finalmente, tenemos también los misteriosos problemas NP, que hasta ahora se han resistido a todos los intentos de clasificarlos de manera precisa.

Cuestiones y ejercicios

1. Suponga que un problema puede resolverse mediante un algoritmo de orden $\Theta(2^n)$. ¿Qué podemos concluir acerca de la complejidad del problema?
2. Suponga que un problema puede resolverse mediante un algoritmo del orden $\Theta(n^2)$, así como mediante otro algoritmo de orden $\Theta(2^n)$. ¿Alguno de los algoritmos será siempre más eficiente que el otro?
3. Enumere todos los subcomités que se pueden formar a partir de un comité compuesto por los dos miembros Alicia y Benito. Enumere todos los subcomités que pueden formarse a partir del comité formado por Alicia, Benito y Carol. ¿Qué sucede con los subcomités que pueden formarse a partir de un comité formado por Alicia, Benito, Carol y David?
4. Proporcione un ejemplo de problema polinómico. Proporcione un ejemplo de problema no polinómino. Proporcione un ejemplo de un problema NP para el que todavía no se haya podido demostrar que es un problema polinómico.
5. Si la complejidad del algoritmo X es mayor que la del algoritmo Y, ¿es el algoritmo X necesariamente más difícil de entender que el algoritmo Y? Explique su respuesta.

Figura 12.12 Resumen gráfico de la clasificación de problemas.



Determinista y no determinista

En muchos casos, existe una línea muy fina entre un "algoritmo" determinista y otro no determinista. Sin embargo, la distinción es bastante clara e importante. Un algoritmo determinista no depende de las características creativas del mecanismo que ejecuta el algoritmo, mientras que un "algoritmo" no determinista sí que podría depender de ellas. Por ejemplo, compare la instrucción

```
Ir hasta la siguiente intersección y girar a la derecha o a la
izquierda.
```

con la instrucción

```
Ir hasta la siguiente intersección y girar a la derecha o a la
izquierda dependiendo de lo que le diga la persona que está de
pie en la esquina.
```

En ambos casos, la acción que lleve a cabo la persona que está siguiendo las instrucciones no está determinada antes de ejecutar realmente la instrucción. Sin embargo, la primera instrucción requiere que la persona que está siguiendo esas instrucciones tome una decisión basándose en su propio criterio y es, por tanto, no determinista. La segunda instrucción no impone ese requisito a la persona que está siguiendo las instrucciones, a esa persona se le dice lo que tiene que hacer en cada etapa. Si varias personas diferentes siguen la primera instrucción, unas girarán hacia la derecha, mientras que otras lo harán hacia la izquierda. Si varias personas siguen la segunda instrucción y reciben la misma información, todas ellas girarán en la misma dirección. Aquí radica una diferencia importante entre los "algoritmos" deterministas y no deterministas. Si un algoritmo determinista se ejecuta repetidamente con los mismos datos de entrada, cada vez se ejecutarán las mismas acciones. Sin embargo, un "algoritmo" no determinista podría dar lugar a acciones diferentes si se repite en condiciones idénticas.

12.6 Criptografía de clave pública

En algunos casos, el hecho de que un problema sea difícil de resolver ha resultado ser una ventaja en lugar de un inconveniente. Un caso especialmente interesante es el de encontrar los factores de un cierto entero, un problema para el que todavía no se ha descubierto una solución eficiente, si es que existe una. Por ejemplo, utilizando únicamente lápiz y papel podríamos comprobar que la tarea de determinar los factores de valores relativamente pequeños, como 2173, es bastante tediosa, y si el número en cuestión fuera tan grande que su representación requiriera varios centenares de dígitos, la tarea sería intratable incluso si aplicáramos tecnología moderna y usáramos las mejores técnicas de factorización actualmente conocidas.

La imposibilidad de determinar una forma eficiente de determinar los factores de números enteros de gran tamaño ha traído de cabeza durante mucho tiempo a numerosos matemáticos, pero en el campo de la criptografía se ha utilizado este hecho para diseñar un método muy popular de cifrado y descifrado de mensajes. Este método se conoce con el nombre de **algoritmo RSA**, un nombre elegido en honor a sus inventores Ron Rivest, Adi Shamir y Len

Adleman. Se trata de un medio de cifrar mensajes utilizando un conjunto de valores conocido con el nombre de **claves de cifrado** y de descifrar dichos mensajes empleando otro conjunto de valores conocido con el nombre de **claves de descifrado**. Las personas que conocen las claves de cifrado pueden cifrar los mensajes, pero no pueden descifrarlos. La única persona que puede descifrar los mensajes es aquella que conozca las claves de descifrado. Por tanto, las claves de cifrado se pueden distribuir a todo el mundo sin problemas y sin comprometer la seguridad del sistema.

Estos sistemas criptográficos se conocen como sistemas de **cifrado de clave pública**, un término que refleja el hecho de que las claves utilizadas para cifrar los mensajes pueden ser de conocimiento público sin que sufra por ello la seguridad del sistema. De hecho, las claves de cifrado se denominan a menudo **claves públicas**, mientras que las claves de descifrado se denominan **claves privadas** (Figura 12.13).

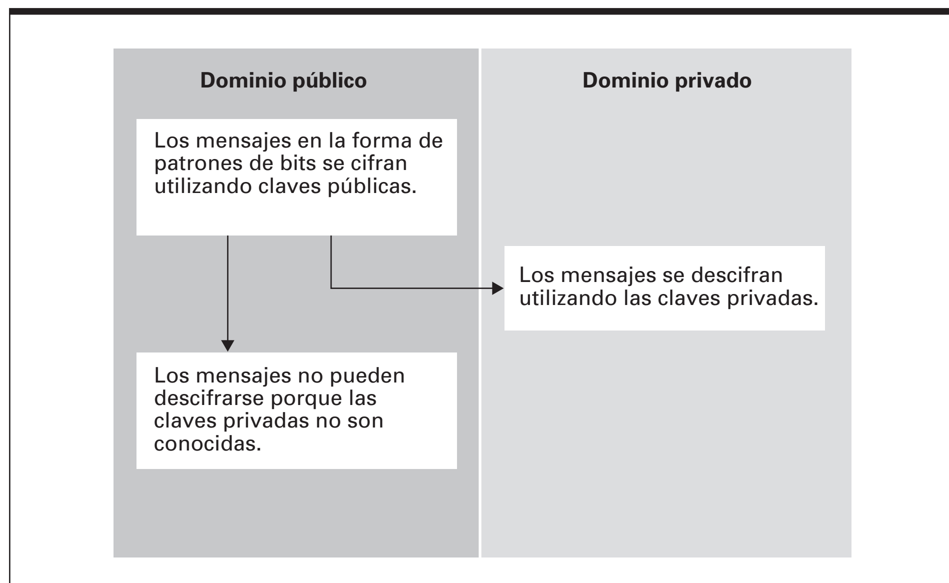
Notación modular

Para describir el sistema de cifrado de clave pública RSA es conveniente utilizar la notación x (mod m), que se lee "x módulo m" o simplemente "x mod m", para representar el resto obtenido al dividir el valor x entre m. Así, 9 (mod 7) es 2 porque 9 ÷ 7 nos da un resto igual a 2. De forma similar, 24 (mod 7) es 3 porque 24 ÷ 7 nos da un resto de 3, y 14 (mod 7) es 0 porque 14 ÷ 7 da un resto igual a 0. Observe que x (mod m) es igual a x si x es un entero comprendido en el rango que va de 0 a $m - 1$. Por ejemplo, 4 (mod 9) es igual a 4.

Las matemáticas nos dicen que si p y q son números primos y m es un entero comprendido entre 0 y pq (el producto de p por q) entonces

$$1 = m^{k(p-1)(q-1)} \pmod{pq}$$

Figura 12.13 Criptografía de clave pública.



para cualquier entero positivo k. Aunque no vamos a demostrar aquí esta igualdad, sí que es conveniente ver un ejemplo con el fin de clarificar qué es lo que queremos decir. Supongamos, por tanto, que p y q son los números primos 3 y 5, respectivamente, y que m es el entero 4. Entonces, esa igualdad afirma que para cualquier entero positivo k, el valor $m^{k(p-1)(q-1)}$ dividido entre 15 (el producto de 3 por 5) nos dará como resto el valor 1. En particular, si $k = 1$, entonces

$$m^{k(p-1)(q-1)} = 4^{1(3-1)(5-1)} = 4^8 = 65.536$$

que al dividirlo entre 15 nos da un resto de 1, tal como establece la igualdad. Además, si $k = 2$, entonces

$$m^{k(p-1)(q-1)} = 4^{2(3-1)(5-1)} = 4^{16} = 4.294.967.296$$

que al dividirlo entre 15 de nuevo nos da como resto 1. De hecho, siempre obtendremos el resto 1, independientemente del entero positivo que seleccionemos para k.

Criptografía de clave pública RSA

Ahora estamos preparados para construir y analizar un sistema de cifrado de clave pública basado en el algoritmo RSA. Primero seleccionamos dos números primos diferentes p y q, cuyo producto representaremos mediante n. A continuación, seleccionamos otros dos enteros positivos e y d tales que $e \times d = k(p - 1)(q - 1) + 1$, para algún entero positivo k. Llamamos a estos valores e y d porque formarán parte del proceso de cifrado (e) y descifrado (d), respectivamente. (El hecho de que se puedan seleccionar valores e y d que satisfagan la ecuación anterior se deduce también de las correspondientes teorías matemáticas, aunque aquí no lo vamos a demostrar.)

Así, hemos seleccionado cinco valores: p, q, n, e y d. Los valores e y n serán las claves de cifrado. Los valores d y n serán las claves de descifrado. Los valores p y q solo se emplean para construir el sistema de cifrado.

> Consideremos un ejemplo específico para clarificar el proceso. Suponga que elegimos 7 y 13 como valores de p y q. Luego $n = 7 \times 13 = 91$. Además, podemos utilizar los valores 5 y 29 para e y d, porque $5 \times 29 = 145 = 144 + 1 = 2(7 - 1)(13 - 1) + 1 = 2(p - 1)(q - 1) + 1$ tal como se requiere. Por tanto, las claves de cifrado serán $n = 91$ y $e = 5$, y las claves de descifrado serán $n = 91$ y $d = 29$. Distribuiremos las claves de cifrado a cualquiera que desee enviarnos mensajes, pero conservaremos para nosotros las claves de descifrado (así como los valores de p y q).

Veamos ahora cómo se cifran los mensajes. Con este objetivo, suponga que un mensaje está codificado en forma de patrón de bits (quizá utilizando ASCII o Unicode) y suponga que el valor de este patrón, cuando se interpreta como representación binaria, es menor que n. (Si no es menor que n, partiríamos el mensaje en segmentos más pequeños y cifraríamos cada segmento individualmente.)

Supongamos que nuestro mensaje, al ser interpretado como representación binaria, representa el valor m. Entonces, la versión cifrada del mensaje será la representación binaria del valor $c = m^e \pmod{n}$. Es decir, el mensaje cifrado será la representación binaria del resto que se obtiene al dividir m^e entre n.

> En particular, si alguien quisiera cifrar el mensaje 10111 utilizando las claves de cifrado $n = 91$ y $e = 5$ que hemos seleccionado en el ejemplo anterior, primero reconocería que 10111 es la representación binaria de 23, luego calcularía $23^e = 23^5 = 6.436.343$, y por último dividiría este valor entre $n = 91$, obteniendo como resto 4. La versión cifrada del mensaje sería por tanto 100, que es la representación binaria de 4.

Para descifrar un mensaje que represente el valor c en notación binaria, calculamos el valor $c^d \pmod{n}$. Es decir, calculamos el valor c^d, dividimos el resultado entre n y nos quedamos con el resto. De hecho, este resto será el valor m del mensaje original porque

$$\begin{aligned} c^d \pmod{n} &= m^{e \times d} \pmod{n} \\ &= m^{k(p-1)(q-1)+1} \pmod{n} \\ &= m \times m^{k(p-1)(q-1)} \pmod{n} \\ &= m \pmod{n} \\ &= m \end{aligned}$$

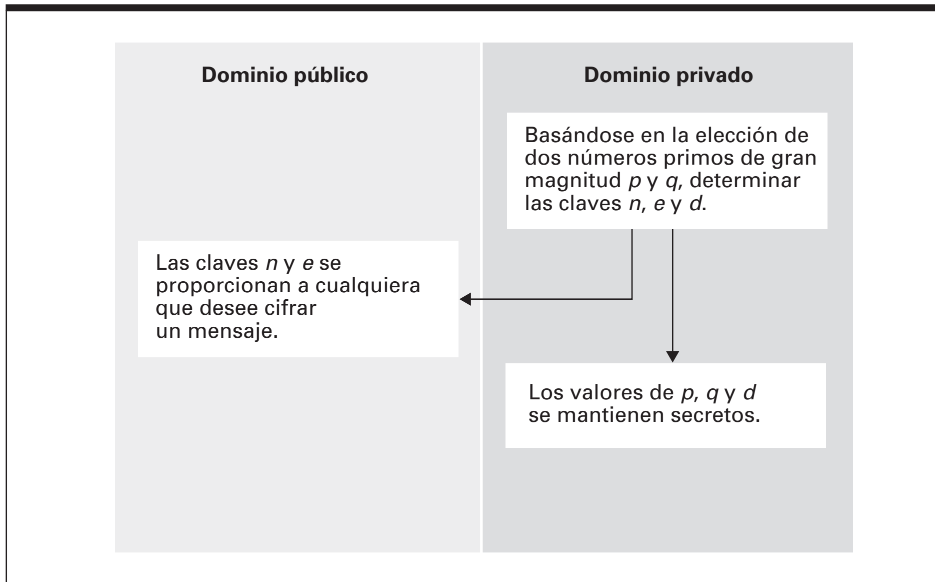
Aquí hemos usado el hecho de que $m^{k(p-1)(q-1)} \pmod{n} = m^{k(p-1)(q-1)} \pmod{pq} = 1$ y también el hecho de que $m \pmod{n} = m$ (porque $m < n$), como hemos indicado anteriormente.

> Continuando con el ejemplo anterior, si recibiéramos el mensaje 100, reconoceríamos que es la representación binaria del valor 4, calcularíamos el valor $4^d = 4^{29} = 288.230.376.151.711.744$, y lo dividiríamos entre $n = 91$ obteniendo como resto el valor 23, que genera el mensaje original 10111 en notación binaria.

En resumen, un sistema de cifrado de clave pública RSA se construye seleccionando dos números enteros primos, p y q, a partir de los cuales generamos los valores n, e y d. Los valores n y e se utilizan para cifrar mensajes y son, por tanto, las claves públicas. Los valores n y d se usan para descifrar los mensajes y son las claves privadas (Figura 12.14). La belleza del sistema radica en que el saber cómo cifrar los mensajes no permite a nadie descifrar esos mismos mensajes. Por tanto, las claves de cifrado n y e pueden ser distribuidas sin ninguna restricción. Si alguien con intenciones maliciosas obtuviera esas claves, seguiría sin ser capaz de descifrar los mensajes interceptados; solo la persona que conozca las claves de descifrado podrá descifrar los mensajes.

La seguridad de este sistema está basada en la suposición de que conocer las claves de cifrado n y e no permite calcular las claves de descifrado n y d. Sin embargo, existen algoritmos que hacen precisamente eso. Una solución consistiría en factorizar el valor n para descubrir los valores p y q, y después determinar d hallando un valor k tal que $k(p - 1)(q - 1) + 1$ sea divisible por e (el

Figura 12.14 Diseño de un sistema de cifrado de clave pública RSA.



cociente sería entonces d). Sin embargo, el primer paso de este proceso puede requerir mucho tiempo, especialmente si los valores de p y q se eligen muy grandes. De hecho, si p y q son tan grandes que sus representaciones binarias requieren cientos de dígitos, entonces los mejores algoritmos de factorización conocidos necesitarían años para poder calcular a partir de n los valores de p y q. Por ello, el contenido de un mensaje cifrado seguiría siendo seguro mucho después de que su importancia se hubiera ya desvanecido.

Hasta la fecha, nadie ha encontrado una forma eficiente de descifrar mensajes basados en la criptografía RSA sin conocer las claves de descifrado, de modo que el cifrado de clave pública basado en el algoritmo RSA se utiliza ampliamente para conseguir confidencialidad a la hora de comunicarse a través de Internet.

Cuestiones y ejercicios

1. Calcule los factores de 66.043. (No pierda mucho tiempo con este problema. Lo importante es que se dé cuenta de que puede hacer falta mucho tiempo.)
2. Utilizando las claves públicas $n = 91$ y $e = 5$, cifre el mensaje 101.
3. Utilizando las claves privadas $n = 91$ y $d = 29$, descifre el mensaje 10.
4. Calcule el valor apropiado para las claves de descifrado n y d en un sistema de criptografía de clave pública RSA basándose en los números primos $p = 7$ y $q = 19$, y utilizando como clave de cifrado $e = 5$.

Problemas de repaso

1. Demuestre cómo puede simularse mediante Bare Bones una estructura de la forma

```
while X equals 0 do;
  .
  .
  .
end;
```

2. Escriba un programa en Bare Bones que almacene un 1 en la variable Z si la variable X es menor o igual que la variable Y, y que almacene un 0 en la variable Z en caso contrario.

3. ¿Cuáles son las principales características de los algoritmos no deterministas?

4. En cada uno de los siguientes casos escriba una secuencia de programa en Bare Bones que realice la actividad indicada.
   a. Asignar 0 a `Z` si el valor de `X` es par; en caso contrario, asignar 1 a `Z`.
   b. Calcular la suma de los enteros comprendidos entre 0 y `X`.

5. Escriba una rutina en Bare Bones que divida el valor de `X` entre el valor de `Y`. Descarte cualquier resto; es decir, 1 dividido entre 2 da 0, y 5 dividido entre 3 da 1.

6. Describa la función computada por el siguiente programa Bare Bones, suponiendo que las entradas de la función están representadas por `X` e `Y` y que su salida está representada por `Z`:

```
copy X to Z;
copy Y to Aux;
while Aux not 0 do;
  decr Z;
  decr Aux;
end;
```

7. Describa la función computada por el siguiente programa Bare Bones, suponiendo que las entradas de la función están representadas por `X` e `Y` y que su salida está representada por `Z`:

```
clear Z;
copy X to Aux1;
copy Y to Aux2;
while Aux1 not 0 do;
  while Aux2 not 0 do;
    decr Z;
    decr Aux2;
  end;
  decr Aux1;
end;
```

8. Escriba un programa Bare Bones que compute la operación OR exclusivo de las variables `X` e `Y`, y que deje el resultado en la variable `Z`. Puede suponer que `X` e `Y` solo pueden comenzar con valores enteros iguales a 0 o a 1.

9. Demuestre que si permitimos etiquetar las sentencias de un programa Bare Bones con valores enteros y sustituimos la estructura de bucle `while` con la bifurcación condicional representada por la forma

```
if nombre not 0 goto etiqueta;
```

donde *nombre* es cualquier variable y ***etiqueta*** es un valor entero utilizado en algún otro lugar para etiquetar una sentencia, entonces el nuevo lenguaje continuará siendo un lenguaje de programación universal.

10. En este capítulo hemos visto cómo puede simularse en Bare Bones la sentencia

```
copy nombre1 to nombre2;
```

Demuestre cómo podría seguir simulándose esa sentencia si sustituimos la estructura de bucle `while` de Bare Bones con un bucle de post-comprobación expresado en la forma

```
repeat ... until (nombre equals 0)
```

11. Demuestre que el lenguaje Bare Bones continuaría siendo un lenguaje universal si sustituimos la sentencia `while` por un bucle de post-comprobación expresado en la forma

```
repeat ... until (nombre equals 0)
```

12. ¿Cuándo rechazaría una máquina de Turing la salida?

13. Diseñe una máquina de Turing que escriba ceros en todas las casillas situadas a la

izquierda de la casilla actual, hasta alcanzar una casilla que contenga un asterisco.

14. Suponga que un patrón de 0s y 1s de la cinta de una máquina de Turing está delimitado mediante asteriscos por ambos extremos. Diseñe una máquina de Turing que rote este patrón una celda hacia la izquierda, asumiendo que cuando la máquina comienza a operar, la casilla actual es el asterisco situado en el extremo derecho del patrón.

15. Diseñe una máquina de Turing que invierta el patrón de 0s y 1s que se encuentre entre la casilla actual (que contiene un asterisco) y el primer asterisco de la izquierda.

16. ¿Qué son los problemas irresolubles?

17. ¿Es auto-terminante el siguiente programa Bare Bones? Explique su respuesta.

```
copy X to Y;
incr Y;
incr Y;
while X not 0 do;
  decr X;
  decr X;
  decr Y;
  decr Y;
end;
decr Y;
while Y not 0 do;
  incr X;
  decr Y;
end;
while X not 0 do;
end;
```

18. ¿Es auto-terminante el siguiente programa Bare Bones? Explique su respuesta.

```
Numero ← 1;
while (Numero <60) do
  (Numero ← Numero + 2)
```

19. ¿Es auto-terminante el siguiente programa Bare Bones? Explique su respuesta.

```
Z ← 1;
X ← 2;
while (X < 50) do
  (Z ←Z+ X;
  X ←X +2)
```

20. Analice la validez de la siguiente pareja de enunciados:

```
El siguiente enunciado es verdadero.
El enunciado anterior es falso.
```

21. Explique la diferencia entre un algoritmo determinista y un algoritmo no determinista.

22. Suponga que se encuentra en un país en el que cada persona es franca o es mentirosa. (Una persona franca siempre dice la verdad, mientras que un mentiroso siempre miente.) ¿Qué pregunta podría hacer a una persona para detectar si es una persona franca o es mentirosa?

23. ¿Cuáles son las principales cuestiones acerca de la complejidad de un problema?

24. Resuma las principales características de la criptografía de clave pública.

25. Suponga que necesitamos averiguar si alguna de las personas de un cierto grupo cumple años en una fecha dada. Una solución sería preguntar a cada una de las personas de una en una. Si adoptamos esta solución, ¿qué suceso sería el que nos dijera que existe tal persona? ¿Qué suceso nos diría que no existe tal persona? Ahora suponga que queremos averiguar si al menos uno de los enteros positivos tiene una propiedad concreta y suponga que aplicamos la misma solución, consistente en ir probando de forma sistemática todos los enteros, de uno en uno. Si fuera cierto que algún entero tiene esa propiedad, ¿cómo lo averiguaríamos? Sin embargo, si ningún entero tuviera esa propiedad, ¿cómo lo averiguaríamos? ¿Cree que la tarea de comprobar si una conjetura es cierta es necesariamente simétrica de la tarea de comprobar si esa conjetura es falsa?

26. ¿Es de tipo polinómico el problema de buscar un cierto valor concreto en una lista? Explique su respuesta.

27. Diseñe un algoritmo para determinar si un cierto entero positivo es primo. ¿Es eficiente su solución? ¿Es una solución de tipo polinómico o no polinómico?

28. ¿Una solución polinómica a un problema es siempre mejor que una solución exponencial? Explique su respuesta.

29. ¿El hecho de que un problema tenga una solución polinómica significa que siempre puede resolverse en un tiempo razonable? Explique su respuesta.

30. Suponga que asignamos al programador Carlos el problema de dividir un grupo (formado por un número par de personas) en dos subgrupos disjuntos de igual tamaño, de manera que la diferencia entre la suma de las edades de cada uno de los subgrupos sea lo mayor posible. Carlos propone la solución de formar todos los posibles pares de subgrupos, calcular la diferencia entre la edad total de cada una de esos pares de subgrupos y seleccionar el par que tenga la máxima diferencia. Por el contrario, la programadora María propone ordenar primero el grupo original por edad y luego dividirlo en dos subgrupos formando uno de los subgrupos a partir de la mitad más joven del grupo ordenado y formando el otro subgrupo a partir de la otra mitad. ¿Cuál es la complejidad de cada una de estas soluciones? ¿Cuál es la complejidad del propio problema, polinómica, NP o no polinómica?

31. ¿Por qué la solución consistente en generar todas las posibles disposiciones de una lista y luego seleccionar aquella que tiene la disposición deseada no es una solución satisfactoria para ordenar una lista?

32. Suponga que un juego de lotería está basado en seleccionar correctamente cuatro valores enteros, cada uno de los cuales está comprendido en el rango que va de 1 hasta 50. Suponga además que el bote llega a ser tan grande que sería beneficioso comprar un billete de lotería para cada una de las posibles combinaciones. Si se tarda un segundo en comprar un único billete, ¿cuánto se tardaría en comprar un billete para cada combinación? ¿Cómo cambiaría el tiempo necesario si esa lotería requiriera seleccionar cinco números en lugar de cuatro? ¿Qué tiene que ver este problema con el material presentado en este capítulo?

33. ¿Es determinista el siguiente algoritmo? Explique su respuesta.

```
procedure cualquierNum (Num)
if (Num == 5)
  then (responder "no")
  else (Num = Num +1
       proporcionar este número
       como respuesta)
```

34. ¿Es determinista el siguiente algoritmo? Explique su respuesta.

```
Tomar un número aleatorio como
entrada.
  Si el número aleatorio es mayor
    que 10, entonces sumarle 20.
  En caso contrario restarle 30.

Tomar otro número aleatorio para
compararlo con el anterior
  Si el primer número es mayor
    entonces intercambiar el valor
    con el otro número.

Escribir los valores de ambos
  números.
```

35. Identifique los puntos no deterministas en el siguiente algoritmo:

```
Seleccionar tres números entre 1 y
  100.
if (la suma de los números
    seleccionados es mayor que 150)
  then (responder "sí")
  else (seleccionar uno de los núme-
        ros elegidos y
        proporcionar ese número
        como respuesta.)
```

36. ¿Qué complejidad temporal tiene el siguiente algoritmo, polinómica o no polinómica? Explique su respuesta.

```
procedure misterio (ListaDeNumeros)
Seleccionar un conjunto de numeros
  de ListaDeNumeros.
if (los números de dicho conjunto
    suman 125)
  then (responder "sí")
  else (no dar una respuesta)
```

37. ¿Cuáles de los siguientes problemas pertenecen a la clase P?

a. Un problema con complejidad n^2
b. Un problema con complejidad $2n$
c. Un problema con complejidad $n^2 + 2n$
d. Un problema con complejidad $n!$

38. Resuma la diferencia entre afirmar que un problema es de tipo polinómico y afirmar que es un problema polinómico no determinista.

39. ¿Qué es un problema intratable? Explique su respuesta.

40. ¿Cuál es la complejidad temporal de cada uno de los siguientes algoritmos?
a. Multiplicación de matrices.
b. Ordenación por burbuja.
c. Búsqueda lineal.
d. Búsqueda binaria.

41. Suponga que tenemos que resolver el problema del viajante para el caso de que existan 15 ciudades y que cada pareja de ciudades está conectada por una carretera distinta. ¿Cuántas rutas diferentes a través de todas las ciudades existirán? ¿Cuánto se tardaría en calcular la longitud de todas esas rutas, suponiendo que la longitud de una ruta puede calcularse en un microsegundo?

42. ¿Cuántas comparaciones entre nombres se realizarán si aplicamos el algoritmo de ordenación por combinación (Figuras 12.9 y 12.8) a la lista formada por los nombres Alicia, Benito, Carol y David? ¿Cuántas se requerirán si la lista está formada por Alicia, Benito, Carol, David y Elena?

43. Explique el uso de una función computable de Turing.

44. Diseñe un algoritmo para determinar soluciones enteras para las ecuaciones de la forma $x^2 + y^2 = n$, donde n es un entero positivo dado. Determine la complejidad temporal del algoritmo.

45. Otro problema que pertenece a la categoría de los problemas NP-completos es el denominado **problema de la mochila**, que es el problema de hallar qué números de una lista tienen una suma igual a un cierto valor concreto.Por ejemplo, los números 257, 388 y 782 son las entradas de la lista

```
642 257 771 388 391 782 304
```

cuya suma es 1427. Determine las entradas cuya suma es igual a 1723. ¿Qué algoritmo ha aplicado? ¿Cuál es la complejidad de ese algoritmo?

46. Identifique las similitudes entre el problema del viajante y el problema de la mochila (véase el Problema 45).

47. El siguiente algoritmo para ordenar listas se denomina ordenación por inserción en la Figura 5.11. ¿Cuántas comparaciones entre entradas de la lista requiere el algoritmo de ordenación por inserción cuando se aplica a una lista de n entradas?

```
procedure OrdenacionInserc (Lista)
N ← 2;
while (el valor de N no exceda la
      longitud de Lista) do
   (Seleccionar la entrada N-ésima de
    Lista como la entrada pivote;
    Mover la entrada pivote a una
    ubicación temporal dejando un
    hueco en Lista;
    while (haya un nombre por encima
    del hueco y dicho nombre sea
    mayor que la entrada pivote)
     do (mover el nombre al hueco
         dejando encima suyo un
         hueco)
         Mover la entrada pivote al
         hueco de Lista;
      N ← N + 1
   )
```

48. Utilice el sistema de cifrado de clave pública RSA para cifrar el mensaje 110 utilizando las claves públicas $n = 91$ y $e = 5$.

49. ¿Cuál es la fortaleza del algoritmo RSA?

50. Suponga que sabemos que las claves públicas de un sistema de cifrado de clave pública basado en el algoritmo RSA son $n = 119$ y $e = 5$. ¿Cuáles serán las claves privadas?

51. ¿Cuándo es auto-terminante un programa Bare Bones?

52. ¿Qué podemos deducir si el entero positivo n no tiene ningún factor entero comprendido en el rango que va de 2 a la raíz cuadrada de n? ¿Qué nos dice esto acerca de la tarea de calcular los factores de un entero positivo?

Cuestiones sociales

Las siguientes cuestiones pretenden ser una guía para los problemas éticos/sociales/legales asociados con el campo de la computación. El objetivo no es responder simplemente a estas cuestiones. El lector debería considerar también por qué las ha contestado de la forma en que lo ha hecho y analizar si sus justificaciones son coherentes entre las distintas cuestiones.

1. Suponga que el mejor algoritmo para resolver un problema requiere 100 años para ejecutarse. ¿Consideraría que ese problema es tratable? ¿Por qué?
2. ¿Deberían los ciudadanos tener el derecho de cifrar sus mensajes de tal forma que se evite su monitorización por parte de organismos gubernamentales? ¿Cree que su respuesta permite imponer "apropiadamente" el cumplimiento de las leyes? ¿Quién debería decidir lo que es una imposición "apropiada" del cumplimiento de las leyes?
3. Si la mente humana es un dispositivo algorítmico, ¿qué consecuencias tiene la tesis de Turing en lo que respecta a la humanidad? ¿Hasta qué grado cree que las máquinas de Turing engloban las capacidades computacionales de la mente humana?
4. Ya hemos visto que existen diferentes modelos computacionales (tablas finitas, fórmulas algebraicas, máquina de Turing, etc.) que tienen diferentes capacidades de computación. ¿Existen diferencias de las capacidades computacionales de los distintos organismos? ¿Existen diferencias en las capacidades computacionales de los distintos seres humanos? En caso afirmativo, ¿deberían los humanos que tienen mayores capacidades poder utilizarlas para conseguir un estilo de vida más acomodado?
5. Hoy día existen sitios web que proporcionan mapas de la mayoría de las ciudades. Estos sitios ofrecen ayuda a la hora de encontrar direcciones concretas, así como capacidades de ampliación para visualizar la disposición de pequeñas partes de una ciudad. Partiendo de esta realidad, considere la siguiente secuencia de suposiciones ficticias. Suponga que esos sitios con mapas incorporaran fotografías de satélite que dispusieran de similares capacidades de ampliación. Suponga que esas capacidades de ampliación se aumentaran para proporcionar una imagen más detallada de los edificios individuales y del paisaje circundante. Suponga que esos servicios de imágenes se mejoraran todavía más incluyendo vídeo en tiempo real. Suponga que esas imágenes de vídeo se mejoraran con tecnología infrarroja. En ese punto, otras personas podrían vernos dentro de nuestra propia casa, 24 horas al día. ¿En qué punto de esta progresión se habrá violado por primera vez nuestro derecho a la intimidad? ¿En qué punto de esta progresión cree que habremos ido más allá de las capacidades de la tecnología actual de los satélites espía? ¿Hasta qué punto es ficticio este escenario?
6. Suponga que una empresa desarrolla y patenta un sistema de cifrado. ¿Debería el gobierno de la nación donde tiene sede esa empresa tener derecho a utilizar el sistema como le venga en gana en nombre de la seguridad nacional? ¿Debería el gobierno de la nación en la que la empresa tiene su sede tener derecho a restringir el uso comercial de ese sistema por parte de

la empresa en nombre de la seguridad nacional? ¿Y qué pasa si la empresa es una multinacional?

7. Suponga que adquiere un producto cuya estructura interna está cifrada. ¿Tiene usted derecho a descifrar la estructura subyacente? En caso afirmativo, ¿tendría derecho a utilizar esa información de manera comercial? ¿Y de manera no comercial? ¿Qué pasa si el cifrado se realizó utilizando un sistema de cifrado secreto y usted descubre el secreto? ¿Tendría derecho a compartir ese secreto?
8. Hace bastantes años, el filósofo John Dewey (1859–1952) introdujo el término "tecnología responsable". Proporcione algunos ejemplos de lo que usted consideraría "tecnología responsable". Basándose en esos ejemplos, formule su propia definición de "tecnología responsable". ¿Cree que la sociedad ha practicado una "tecnología responsable" a lo largo de los últimos 100 años? ¿Debería llevarse a cabo algún tiempo de acción para garantizar que lo haga? En caso afirmativo, ¿qué acciones serían esas?; en caso contrario, ¿por qué no?

Lecturas adicionales

Garey, M. R. y D. S. Johnson. ***Computers and Intractability.*** Nueva York: W. H. Freeman, 1979.

Hamburger, H. y D. Richards. ***Logic and Language Models for Computer Science***. Englewood Cliffs, NJ: Prentice-Hall, 2002.

Hofstadter, D. R. ***Gödel, Escher, Bach: An Eternal Golden Braid.*** St. Paul, MN: Vintage, 1980.

Hopcroft, J. E., R. Motwani y J. D. Ullman. ***Introduction to Automata Theory, Languages, and Computation***, 3ª ed. Boston, MA: Addison-Wesley, 2007.

Lewis, H. R. y C. H. Papadimitriou. ***Elements of the Theory of Computation***, 2ª ed. Englewood Cliffs, NJ: Prentice-Hall, 1998.

Rich E. Automata. ***Computability, and Complexity: Theory and Application.*** Upper Saddle River, NJ: Prentice-Hall, 2008.

Sipser, M. ***Introduction to the Theory of Computation.*** Boston: PWS, 1996.

Smith, C. y E. Kinber. ***Theory of Computing: A Gentle Introduction.*** Englewood Cliffs, NJ: Prentice-Hall, 2001.

Sudkamp, T. A. ***Languages and Machines: An Introduction to the Theory of Computer Science***, 3ª ed. Boston, MA: Addison-Wesley, 2006.

Apéndices



apéndice

A

ASCII

La siguiente tabla proporciona una lista parcial del código ASCII. A cada patrón de bits se le ha añadido un 0 a la izquierda con el fin de obtener el patrón de 8 bits comúnmente utilizado hoy día. En la tercera columna se especifica el valor hexadecimal de cada patrón de 8 bits.

Símbolo	ASCII	Hex	Símbolo	ASCII	Hex	Símbolo	ASCII	Hex
avance de línea	00001010	0A	>	00111110	3E	^	01011110	5E
retorno de carro	00001011	0B	?	00111111	3F	_	01011111	5F
espacio	00100000	20	@	01000000	40	`	01100000	60
!	00100001	21	A	01000001	41	a	01100001	61
”	00100010	22	B	01000010	42	b	01100010	62
#	00100011	23	C	01000011	43	c	01100011	63
$	00100100	24	D	01000100	44	d	01100100	64
%	00100101	25	E	01000101	45	e	01100101	65
&	00100110	26	F	01000110	46	f	01100110	66
’	00100111	27	G	01000111	47	g	01100111	67
(	00101000	28	H	01001000	48	h	01101000	68
)	00101001	29	I	01001001	49	i	01101001	69
*	00101010	2A	J	01001010	4A	j	01101010	6A
+	00101011	2B	K	01001011	4B	k	01101011	6B
’	00101100	2C	L	01001100	4C	l	01101100	6C
-	00101101	2D	M	01001101	4D	m	01101101	6D
.	00101110	2E	N	01001110	4E	n	01101110	6E
/	00111111	2F	O	01001111	4F	o	01101111	6F
0	00110000	30	P	01010000	50	p	01110000	70
1	00110001	31	Q	01010001	51	q	01110001	71
2	00110010	32	R	01010010	52	r	01110010	72
3	00110011	33	S	01010011	53	s	01110011	73
4	00110100	34	T	01010100	54	t	01110100	74
5	00110101	35	U	01010101	55	u	01110101	75
6	00110110	36	V	01010110	56	v	01110110	76
7	00110111	37	W	01010111	57	w	01110111	77
8	00111000	38	X	01011000	58	x	01111000	78
9	00111001	39	Y	01011001	59	y	01111001	79
:	00111010	3A	Z	01011010	5A	z	01111010	7A
;	00111011	3B	[	01011011	5B	{	01111011	7B
<	00111100	3C	\	01011100	5C	|	01111100	7C
=	00111101	3D	]	01011101	5D	}	01111101	7D

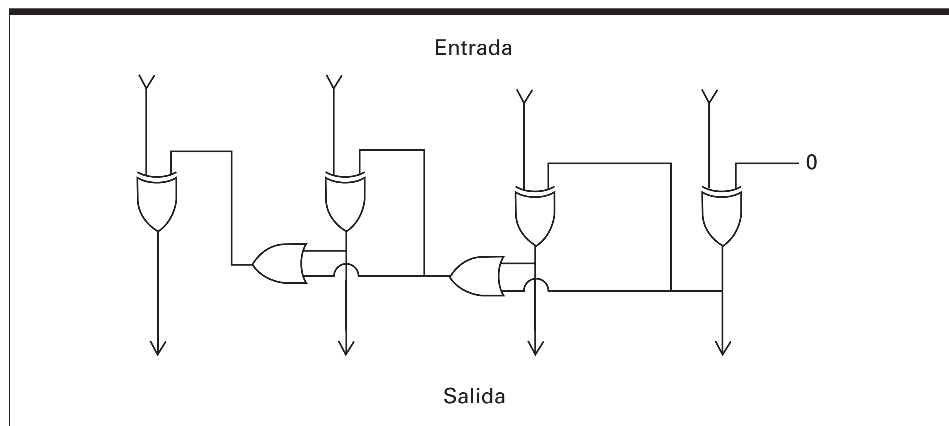
apéndice

B

Circuitos para manipular representaciones en complemento a dos

Este apéndice presenta una serie de circuitos para invertir y sumar valores representados en notación de complemento a dos. Comenzaremos con el circuito de la Figura B.1 que convierte una representación en complemento a dos de cuatro bits a la representación correspondiente al negado de dicho valor. Por ejemplo, dada la representación en complemento a dos de 3, el circuito genera la representación de −3. El circuito lleva a cabo la tarea siguiendo el mismo algoritmo que se presenta en el texto. Es decir, copia el patrón de derecha a izquierda hasta copiar el primer 1 y luego complementa cada uno de lo bits restantes a medida que los pasa de la entrada a la salida. Puesto que una de las entradas de la puerta XOR situada más a la derecha está fija a 0, esta puerta se limitará a pasar su otra entrada a la salida. Sin embargo, esta salida también se pasa hacia la izquierda, como una de las entradas de la siguiente puerta XOR. Si esta salida es 1, la siguiente puerta XOR complementará su bit de entrada al pasarlo a la salida. Además, este 1 también pasará hacia la izquierda a través de la puerta OR, con el fin de afectar a la siguiente puerta. De esta manera, el primer 1 que se copie a la salida pasará también hacia la izquierda, haciendo que

Figura B.1 Un circuito que calcula el inverso de un patrón en complemento a dos.



todos los bits restantes queden complementados a medida que pasan hacia la salida.

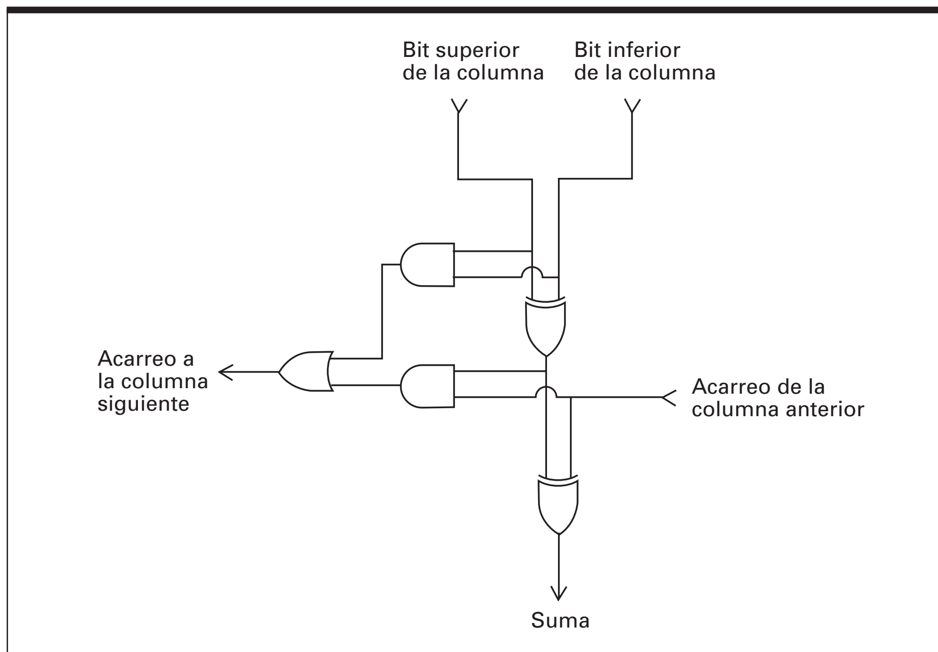
A continuación, consideramos el proceso de sumar dos valores representados en notación de complemento a dos. En particular, a la hora de resolver el problema

```
+ 0110
+ 1011
```

iremos columna a columna de derecha a izquierda ejecutando el mismo algoritmo para cada columna. Así, una vez que obtengamos un circuito para sumar una columna en este tipo de problema, podremos construir un circuito que sume varias columnas repitiendo simplemente el circuito de una sola columna.

El algoritmo para sumar una única columna en un problema de suma de varias columnas consiste en sumar los dos valores de la columna actual, añadirle a esa suma cualquier posible acarreo de la columna anterior, escribir el bit menos significativo de esta suma en la respuesta y transferir cualquier acarreo que se haya producido hacia la siguiente columna. El circuito de la Figura B.2 sigue precisamente este algoritmo. La puerta XOR superior determina la suma de los dos bits de entrada. La puerta XOR inferior añade a esta suma el valor del acarreo procedente de la columna anterior. Las dos puertas AND junto con la puerta OR pasan hacia la izquierda cualquier acarreo que se haya producido en el proceso. En particular, se generará un acarreo de salida igual a 1 si los dos bits de entrada originales de esta columna eran 1 o si la suma de estos bits es 1 y el acarreo de entrada también es 1.

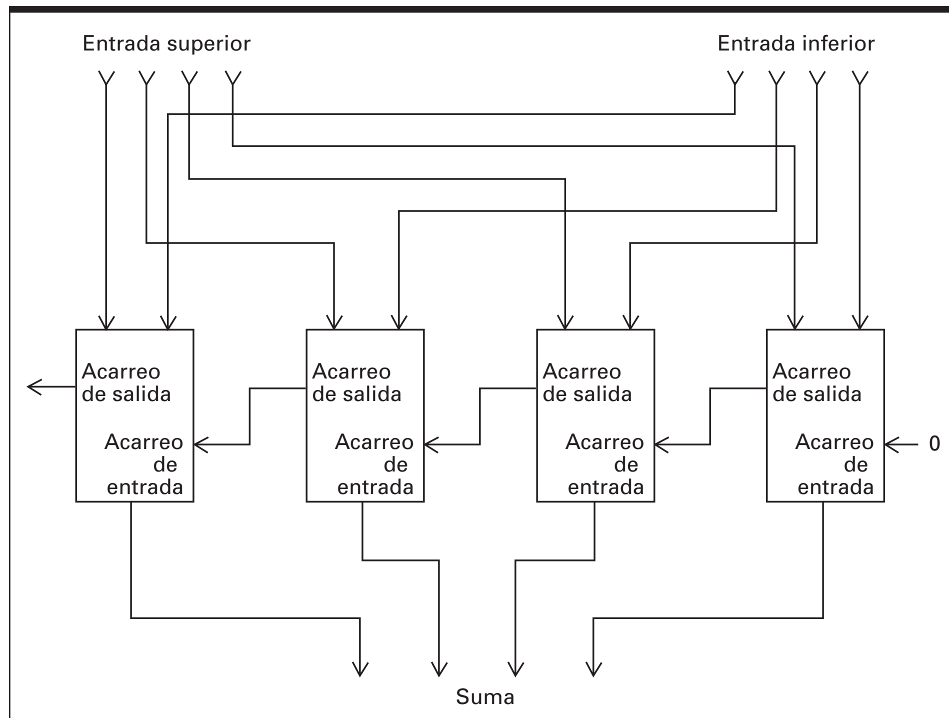
Figura B.2 Un circuito para sumar una única columna en un problema de suma de varias columnas.



La Figura B.3 muestra cómo pueden utilizarse copias de este circuito de una única columna para componer un circuito que calcule la suma de dos valores representados en un sistema de complemento a dos de cuatro bits. Cada rectángulo representa una copia del circuito de suma de una única columna. Observe que el valor de acarreo que se le entrega al rectángulo situado más a la derecha es siempre 0, porque no existe ningún acarreo de ninguna columna anterior. De forma similar, el acarreo generado por el rectángulo generado más a la izquierda se ignora.

El circuito de la Figura B.3 se conoce con el nombre de ***sumador en cascada*** porque la información de acarreo debe propagarse, o transmitirse en cascada, desde la columna situada más a la derecha hasta la columna situada más a la izquierda. Aunque su composición es bastante simple, estos circuitos son más lentos a la hora de realizar su función que otras versiones más inteligentes, como por ejemplo el sumador de acarreo anticipado, que minimiza esta propagación entre columnas. Por tanto, el circuito de la Figura B.3, aunque resulta suficiente para nuestros propósitos, no es el circuito utilizado en las máquinas de hoy en día.

Figura B.3 Un circuito para sumar dos valores en notación de complemento a dos utilizando cuatro copias del circuito de la Figura B.2





Un lenguaje máquina simple

En este apéndice presentamos un lenguaje máquina simple pero representativo. Comenzaremos explicando la arquitectura de la propia máquina.

La arquitectura de la máquina

La máquina tiene 16 registros de propósito general numerados de 0 a F (en hexadecimal). Cada registro tiene una longitud de un byte (ocho bits). Para identificar los registros dentro de las instrucciones, a cada registro se le asigna el patrón de cuatro bits distintivo que representa su número de registro. Así, el registro 0 se identifica mediante 0000 (0 en hexadecimal), mientras que el registro 4 se identifica mediante 0100 (4 en hexadecimal).

Hay 256 celdas en la memoria principal de la máquina. Cada celda tiene asignada una dirección unívoca consistente en un número entero comprendido en el rango que va de 0 a 255. Por tanto, una dirección puede representarse mediante un patrón de ocho bits que va de 00000000 a 11111111 (o un valor hexadecimal en el rango comprendido entre 00 y FF).

Se supone que los valores en punto flotante están almacenados en el formato de ocho bits que se ha explicado en la Sección 1.7 y resumido en la Figura 1.26.

El lenguaje de la máquina

Cada instrucción del lenguaje máquina tiene dos bytes de longitud. Los primeros 4 bits proporcionan el código de operación, mientras que los 12 bits restantes forman el campo de operando. La siguiente tabla enumera las instrucciones en notación hexadecimal, proporcionando una breve descripción de cada una de ellas. Se utilizan las letras R, S y T, en lugar de dígitos hexadecimales, en aquellos campos que representen un identificador de registro que varía dependiendo de la aplicación concreta de la instrucción. Las letras X e Y se emplean, en lugar de dígitos hexadecimales, en aquellos campos variables que no representen un registro.

Código operando	Operando	Descripción
1	RXY	Cargar (LOAD) el registro R con el patrón de bits que se encuentra en la celda de memoria cuya dirección es XY. ***Ejemplo***: 14A3 haría que el contenido de la celda de memoria ubicada en la dirección A3 se almacenara en el registro 4.
2	RXY	Cargar (LOAD) el registro R con el patrón de bits XY. ***Ejemplo***: 20A3 haría que se almacenara el valor A3 en el registro 0.
3	RXY	Almacenar (STORE) en la celda de memoria cuya dirección es XY el patrón de bits contenido en el registro R. ***Ejemplo***: 35B1 haría que el contenido del registro 5 se almacenara en la celda de memoria cuya dirección es B1.
4	0RS	Copiar (MOVE) al registro S el patrón de bits contenido en el registro R. ***Ejemplo***: 40A4 haría que el contenido del registro A se copiara en el registro 4.
5	RST	Sumar (ADD) los patrones de bits de los registros S y T como si fueran representaciones en complemento a dos y almacenar el resultado en el registro R. ***Ejemplo***: 5726 haría que se sumaran los valores binarios de los registros 2 y 6, y que el resultado de la suma se almacenara en el registro 7.
6	RST	Sumar (ADD) los patrones de bits de los registros S y T como si fueran representaciones en notación de punto flotante y almacenar el resultado en el registro R. ***Ejemplo***: 634E haría que se sumaran los valores de los registros 4 y E como valores en punto flotante y que el resultado se almacenara en el registro 3.
7	RST	Combinar mediante OR los patrones de bits de los registros S y T y almacenar el resultado en el registro R. ***Ejemplo***: 7CB4 haría que el resultado de combinar mediante OR el contenido de los registros B y 4 se almacenara en el registro C.
8	RST	Combinar mediante AND los patrones de bits de los registros S y T y almacenar el resultado en el registro R. ***Ejemplo***: 8045 haría que el resultado de combinar mediante AND el contenido de los registros 4 y 5 se almacenara en el registro 0.
9	RST	Combinar mediante EXCLUSIVE OR los patrones de bits de los registros S y T y almacenar el resultado en el registro R. ***Ejemplo***: 95F3 haría que el resultado de combinar mediante la operación EXCLUSIVE OR el contenido de los registros F y 3 se almacenara en el registro 5.
A	R0X	Rotar (ROTATE) X bits hacia la derecha el patrón de bits contenido en el registro R. Los bits que salen por el extremo de menor peso se vuelven a insertar en el extremo de mayor peso. ***Ejemplo***: A403 haría que el contenido del registro 4 se rotara 3 bits hacia la derecha, de forma circular.
B	RXY	Saltar (JUMP) a la instrucción ubicada en la celda de memoria correspondiente a la dirección XY si el patrón de bits contenido en el registro R es igual al patrón de bits contenido en el registro número 0. En caso contrario, continuar con la secuencia normal de ejecución. (El salto se implementa copiando XY en el contador de programa durante la fase de ejecución.) ***Ejemplo***: B43C compararía primero el contenido del registro 4 con el del registro 0. Si los dos fueran iguales, colocaría el patrón 3C en el contador de programa, de modo que la siguiente instrucción que se ejecute sea la que está ubicada en la siguiente dirección de memoria. En caso contrario, no se haría nada y la ejecución del programa continuaría con su secuencia normal.
C	000	Detener (HALT) la ejecución. ***Ejemplo***: C000 haría que se detuviera la ejecución del programa.

apéndice

D

Lenguajes de programación de alto nivel

Este apéndice proporciona un breve resumen sobre cada uno de los lenguajes utilizados como ejemplos en el Capítulo 6.

Ada

El lenguaje Ada, denominado así en honor de Augusta Ada Byron (1815–1851), que fue una defensora de Charles Babbage y la hija del poeta Lord Byron, fue desarrollado por iniciativa del Departamento de Defensa de Estados Unidos en un intento de obtener un único lenguaje de propósito general que satisfaciera todas sus necesidades de desarrollo de software. Uno de los principales objetivos durante el diseño de Ada fue incorporar características para programar sistemas de computadora en tiempo real que pudieran utilizarse como parte de máquinas de mayor tamaño, como por ejemplo sistemas de guiado de misiles, sistemas de control medioambiental dentro de edificios y sistemas de control para automóviles y pequeños electrodomésticos. Ada contiene por tanto características para expresar actividades en entornos de procesamiento paralelo, además de una serie de técnicas de utilidad para la gestión de casos especiales (denominados excepciones) que pueden surgir dentro del entorno de aplicación. Aunque originalmente se desarrolló como un lenguaje imperativo, las versiones más recientes de Ada han adoptado el paradigma de la orientación a objetos.

El diseño del lenguaje Ada ha puesto siempre el énfasis en aquellas características que permiten un desarrollo eficiente de software fiable, una característica que queda ilustrada por el hecho de que todo el software de control interno del avión Boeing 777 fue escrito en Ada. Esta es también una de las principales razones por las que se utilizó Ada como punto de partida a la hora de desarrollar el lenguaje SPARK, como se ha indicado en el Capítulo 5.

C

El lenguaje C fue desarrollado por Dennis Ritchie en los Bell Laboratories a principios de la década de 1970. Aunque diseñado originalmente como un lenguaje para el desarrollo de software de sistemas, C ha conseguido una gran

popularidad en toda la comunidad de programadores y ha sido estandarizado por el instituto ANSI.

C se concibió originalmente como un lenguaje situado un escalón por encima del lenguaje máquina. En consecuencia, su sintaxis es muy seca, si la comparamos con otros lenguajes de alto nivel que emplean palabras en inglés completas para expresar algunas de las primitivas que en C se representan mediante símbolos especiales. Esta sequedad permite una representación eficiente de algoritmos complejos, lo cual es una de las principales razones de la popularidad del lenguaje C. (A menudo, una representación concisa es más legible que otra muy larga.)

C++

El lenguaje C++ fue desarrollado por Bjarne Stroustrup en los Bell Laboratories como una versión ampliada del lenguaje C. El objetivo era obtener un lenguaje compatible con el paradigma de la orientación a objetos. Actualmente, C++ no es solo uno de los principales lenguajes orientados a objetos, sino que se ha utilizado como punto de partida para el desarrollo de otros dos de los principales lenguajes orientados a objetos: Java y C#.

C#

El lenguaje C# fue desarrollado por Microsoft como una herramienta para el entorno .NET, que es un sistema muy completo para el desarrollo de software de aplicación para máquinas que ejecuten el software de sistemas de Microsoft. El lenguaje C# es muy similar a C++ y Java. De hecho, la razón de que Microsoft introdujera C# como un lenguaje diferente, no es porque se trate de un lenguaje verdaderamente nuevo, sino que como lenguaje diferente que es, Microsoft podía personalizar características específicas del lenguaje sin preocuparse de cumplir con los estándares que afectaban a otros lenguajes y sin preocuparse tampoco por los derechos de propiedad intelectual de otras empresas. Por tanto, la novedad de C# radica en su papel como lenguaje principal para el desarrollo de software que utilice el entorno .NET. Con el respaldo de Microsoft, C# y el entorno .NET prometen ser actores relevantes en el campo del desarrollo software durante los años venideros.

Fortran

FORTRAN es el acrónimo de FORmula TRANslator (traductor de fórmulas). Este lenguaje fue uno de los primeros lenguajes de alto nivel que se desarrollaron (fue presentado en 1957) y fue también uno de los primeros lenguajes en obtener una amplia aceptación dentro de la comunidad de los profesionales de la computación. A lo largo de los años, su descripción oficial ha sufrido numerosas ampliaciones, lo que significa que el lenguaje FORTRAN actual es muy diferente del original. De hecho, estudiando la evolución de FORTRAN podemos ver los efectos que las sucesivas investigaciones han ido teniendo en el diseño de los lenguajes de programación. Aunque originalmente se diseñó como lenguaje imperativo, las versiones más recientes de FORTRAN incluyen ahora muchas características de orientación a objetos. FORTRAN continúa siendo un lenguaje muy popular dentro de la comunidad científica. En particu-

lar, muchos paquetes estadísticos y de análisis numérico están escritos en FORTRAN y muy probablemente sigan estándolo en el futuro.

Java

Java es un lenguaje orientado a objetos desarrollado por Sun Microsystems a principios de la década de 1990. Sus diseñadores tomaron prestadas numerosas ideas de C y C++. El entusiasmo por Java se debe no al propio lenguaje, sino a su implementación universal y al impresionante número de plantillas prediseñadas que hay disponibles en el entorno de programación Java. Lo de implementación universal significa que un programa escrito en Java puede ejecutarse de forma eficiente en un amplio rango de máquinas, mientras que la disponibilidad de plantillas significa que puede desarrollarse software complejo con relativa facilidad. Por ejemplo, las plantillas tipo applet y servlet permiten simplificar el desarrollo de software para la World Wide Web.

apéndice

E

Equivalencia de las estructuras iterativas y recursivas

En este apéndice vamos a utilizar el lenguaje de Bare Bones del Capítulo 11 como herramienta para responder a la cuestión planteada en el Capítulo 4 en relación con la potencia de las estructuras iterativas y recursivas. Recuerde que Bare Bones contiene solo tres sentencias de asignación (`clear`, `incr` y `decr`) y una estructura de control (construida a partir de una pareja de sentencias `while-end`). Este lenguaje tan simple tiene la misma potencia de computación que una máquina de Turing; por tanto, si aceptamos la tesis de Church-Turing, podemos concluir que todo problema con una solución algorítmica tendrá una solución expresable en Bare Bones.

El primer paso en la comparación de las estructuras iterativas y recursivas consiste en sustituir la estructura iterativa de Bare Bones por una estructura recursiva. Haremos esto eliminando las sentencias `while` y `end` del lenguaje y proporcionando, en su lugar, la capacidad de dividir un programa Bare Bones en diferentes unidades, con la posibilidad de invocar una de esas unidades desde otro punto del programa. De manera más precisa, lo que proponemos es que cada programa en el lenguaje modificado consista en una serie de unidades de programa sintácticamente disjuntas. Vamos a suponer que cada programa tiene que contener exactamente un unidad denominada `MAIN` que tenga la siguiente estructura sintáctica

```
MAIN: begin;
        .
        .
        .
      end;
```

(donde los puntos representan otras sentencias Bare Bones) y quizá otras unidades (semánticamente subordinadas a `MAIN`) que tendrán la estructura:

```
unidad: begin;
    .
    .
    .
return;
```

(donde *unidad* representa el nombre de la unidad que tiene la misma sintaxis que los nombres de variable). La semántica de esta estructura particionada es que el programa siempre inicia su ejecución al principio de la unidad `MAIN` y se detiene al alcanzar la sentencia final de dicha unidad. Las otras unidades de programa distintas de `MAIN` pueden invocarse como procedimientos por medio de la sentencia condicional

```
if nombre not 0 perform unidad;
```

(donde *nombre* representa cualquier nombre de variable y *unidad* representa cualquiera de los nombres de unidades de programa diferentes de `MAIN`). Además, permitiremos que las otras unidades diferentes de `MAIN` se invoquen de manera recursiva.

Con estas características añadidas, podemos simular la estructura `while-end` incluida en el lenguaje Bare Bones original. Por ejemplo, un programa Bare Bones de la forma

```
while X not 0 do;
S;
end;
```

(donde `S` representa cualquier secuencia de sentencias Bare Bones) puede sustituirse por la siguiente estructura de unidad

```
MAIN: begin;
         if X not 0 perform unidadA;
      end;
unidadA: begin;
         S;
         if X not 0 perform unidadA;
      return;
```

En consecuencia, podemos concluir que el lenguaje modificado tiene todas las capacidades del lenguaje Bare Bones original.

También se puede demostrar que cualquier problema que pueda resolverse utilizando el lenguaje modificado, puede resolverse empleando Bare Bones. Un método de hacer esto consiste en mostrar cómo cualquier algoritmo expresado en el lenguaje modificado podría escribirse también en el lenguaje Bare Bones original. Sin embargo, esto implica una descripción explícita de cómo simular estructuras recursivas mediante la estructura `while-end` de Bare Bones.

Para nuestros propósitos, resulta más simple recurrir a la tesis de Church-Turing, tal como la hemos presentado en el Capítulo 11. En particular, la tesis de Church-Turing combinada con el hecho de que Bare Bones tiene la misma potencia que las máquinas de Turing, indica que no hay ningún lenguaje que pueda ser más potente que nuestro lenguaje Bare Bones original. Por tanto,

cualquier problema resoluble en nuestro lenguaje modificado puede también resolverse utilizando Bare Bones.

Concluimos así que la potencia del lenguaje modificado coincide con la del lenguaje Bare Bones original. La única diferencia entre los dos lenguajes es que uno de ellos proporciona una estructura de control iterativa y el otro proporciona un mecanismo de recursión. Por tanto, las dos estructuras de control son equivalentes, de hecho, en términos de potencia de computación.

apéndice

F

Respuestas a las cuestiones y ejercicios

Capítulo 1

Sección 1.1

1. Una y solo una de las dos entradas superiores tiene que ser 1 y la entrada inferior tiene que ser también 1.
2. El 1 en la entrada inferior es transformado en 0 por la puerta NOT, haciendo que la salida de la puerta AND sea 0. Por tanto, ambas entradas de la puerta OR son 0 (recuerde que la entrada superior del biestable se mantiene a 0), por lo que la salida de la puerta OR pasa a ser 0. Esto significa que la salida de la puerta AND continuará teniendo el valor 0 después de que la entrada inferior del biestable vuelva a 0.
3. La salida de la puerta OR superior estará a 1, haciendo que la puerta NOT superior genere una salida igual a 0. Esto hará que la puerta OR inferior genere un 0, haciendo que la puerta NOT inferior tenga una salida igual a 1. Este 1 es el que se ve como salida del biestable, además de ser realimentado hacia la puerta OR superior, donde mantiene la salida de dicha puerta en el nivel 1, incluso después de que la entrada del biestable haya vuelto a 0.
4. **a.** El circuito completo es equivalente a una única puerta OR.
   b. Este circuito completo también es equivalente a una única puerta XOR.
5. **a.** 6AF2 **b.** E85517 **c.** 48
6. **a.** 0101111110110010111
   b. 0110000100001010
   c. 1010101111001101
   d. 0000000100000000

Sección 1.2

1. En el primer caso, la celda de memoria número 6 termina conteniendo el valor 5. En el segundo caso, termina conteniendo el valor 8.

2. El Paso 1 borra el valor original contenido en la celda número 3, al escribirse allí el nuevo valor. En consecuencia, el Paso 2 no almacena el valor original de la celda número 3 en la celda número 2. El resultado es que ambas celdas terminan teniendo el valor que estaba originalmente almacenado en la celda número 2. Un procedimiento correcto sería el siguiente:

   Paso 1. Desplazar el contenido de la celda número 2 a la celda número 1.

   Paso 2. Desplazar el contenido de la celda número 3 a la celda número 2.

   Paso 3. Desplazar el contenido de la celda número 1 a la celda número 3.
3. 32768 bits.

Sección 1.3

1. Una extracción más rápida de los datos y tasas de transferencia más altas.
2. El punto que hay que recordar es que la lentitud del movimiento mecánico, comparada con la velocidad de funcionamiento interno de la computadora, dicta que debemos minimizar el número de veces que haya que desplazar los cabezales de lectura/escritura. Si rellenamos una superficie completa antes de comenzar con la siguiente, debemos desplazar el cabezal de lectura/escritura cada vez que terminemos con una pista. El número de desplazamientos será, por tanto, aproximadamente igual al número de total de pistas de las dos superficies. Sin embargo, si alternamos entre una superficie y otra conmutando electrónicamente entre los cabezales de lectura/escritura, solo tendremos que desplazar los cabezales de lectura/escritura después de haber rellenado cada cilindro.
3. En esta aplicación, la información debe extraerse del almacenamiento masivo de forma aleatoria, lo que requeriría mucho tiempo si estamos utilizando el sistema en espiral que se emplea en los discos CD y DVD. Además, la tecnología actual no permite actualizar porciones individuales de los datos en un CD o un DVD.
4. El espacio de almacenamiento se asigna en unidades de sectores físicos (en realidad, en unidades de grupos de sectores, en la mayoría de los casos). Si el último sector físico no está lleno, puede añadirse texto adicional sin incrementar el espacio de almacenamiento asignado al archivo. Si el último sector físico está lleno, cualquier nueva adición al documento requerirá la asignación de sectores físicos adicionales.
5. Las unidades flash no requieren que se efectúe ningún movimiento físico, por lo que tienen tiempos de respuesta más cortos y no sufren desgaste físico.
6. Un buffer es un área de almacenamiento de datos utilizada para almacenar datos de manera temporal, usualmente como forma de absorber las incoherencias existentes entre el origen de los datos y su destino final.

Sección 1.4

1. Ciencias de la computación.

2. Los dos patrones son iguales, salvo por el hecho de que el sexto bit comenzando por el extremo de menor peso es siempre igual a 0 para las letras mayúsculas y a 1 para la minúsculas.

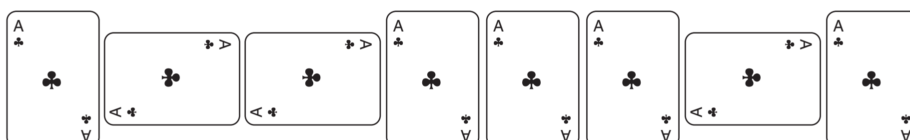
3. **a.**

00100010	01010011	01110100	01101111
01110000	00100001	00100010	00100000
01000011	01101000	01100101	01110010
01111001	01101100	00100000	01110011
01101000	01101111	01110101	01110100
01100101	01110100	00101110	

b.

01000100	01101111	01100101	01110011
00100000	00110010	00100000	00101011
00100000	00110011	00100000	00111101
00100000	00110101	00111111	

4.



5. **a.** 5 **b.** 9 **c.** 11 **d.** 6 **e.** 16 **f.** 18

6. **a.** 110 **b.** 1101 **c.** 1011 **d.** 10010 **e.** 11011 **f.** 100

7. En 24 bits, podemos almacenar tres símbolos utilizando ASCII. Por tanto, podemos almacenar valores numéricos hasta 999. Sin embargo, si usamos los bits como dígitos binarios podemos almacenar valores hasta 16.777.215.

8. **a.** 15.15 **b.** 51.0.128 **c.** 10.160

9. Las representaciones geométricas permiten realizar más fácilmente los cambios de escala que las imágenes codificadas en forma de mapa de bits. Sin embargo, las representaciones geométricas no proporcionan normalmente la misma calidad fotográfica que los mapas de bits. De hecho, como se explica en la Sección 1.8, las representaciones JPEG de mapas de bits son muy populares en el campo de la fotografía.

10. Con una tasa de muestreo de 44.100 muestras por segundo, una hora de música estéreo requeriría 635.040.000 bytes de almacenamiento. Por tanto, se llenaría aproximadamente un CD, cuya capacidad es ligeramente mayor que 600MB.

Sección 1.5

1. **a.** 42 **b.** 33 **c.** 23 **d.** 6 **e.** 31

2. **a.** 100000 **b.** 1000000 **c.** 1100000 **d.** 1111 **e.** 11011

3. **a.** $3\frac{1}{4}$ **b.** $5\frac{7}{8}$ **c.** $2\frac{1}{2}$ **d.** $6\frac{3}{8}$ **e.** $\frac{5}{8}$

4. **a.** 100.1 **b.** 10.11 **c.** 1.001 **d.** 0.0101 **e.** 101.101

5. **a.** 100111 **b.** 1011.110 **c.** 100000 **d.** 1000.00

Sección 1.6

1. **a.** 3 **b.** 15 **c.** −4 **d.** −6 **e.** 0 **f.** −16

2. **a.** 00000110 **b.** 11111010 **c.** 11101111
   d. 00001101 **e.** 11111111 **f.** 00000000

3. **a.** 11111111 **b.** 10101011 **c.** 00000100
   d. 00000010 **e.** 00000000 **f.** 10000001

4. **a.** Con 4 bits el valor mayor es 7 y el más pequeño es −8.
   b. Con 6 bits el valor mayor es 31 y el más pequeño es −32.
   c. Con 8 bits el valor mayor es 127 y el más pequeño es −128.

5. **a.** 0111 (5 + 2 = 7) **b.** 0100 (3 + 1 = 4) **c.** 1111 (5 + (−6) = −1)
   d. 0001 (−2 + 3 = 1) **e.** 1000 (−6 + (−2) = −8)

6. **a.** 0111 **b.** 1011 (desbordamiento) **c.** 0100 (desbordamiento)
   d. 0001 **e.** 1000 (desbordamiento)

7.

a.	**b.**	**c.**	**d.**	**e.**
0110	0011	0100	0010	0001
+ 0001	+1110	+ 1010	+ 0100	+1011
0111	0001	1110	0110	1100

8. No. El desbordamiento tiene lugar cuando se intenta almacenar un número demasiado grande para el sistema que se está utilizando. Cuando se suma un valor positivo con un valor negativo, el resultado debe encontrarse entre los dos valores que se están sumando. Por tanto, el resultado será lo suficientemente pequeño como para poder almacenarlo sin ningún error.

9. **a.** 6 porque 1110 → 14 − 8
   b. −1 porque 0111 → 7 − 8
   c. 0 porque 1000 → 8 − 8
   d. −6 porque 0010 → 2 − 8
   e. −8 porque 0000 → 0 − 8
   f. 1 porque 1001 → 9 − 8

10. **a.** 1101 porque 5 + 8 = 13 → 1101
    b. 0011 porque −5 + 8 = 3 → 0011
    c. 1011 porque 3 + 8 = 11 → 1011
    d. 1000 porque 0 + 8 = 8 → 1000
    e. 1111 porque 7 + 8 = 15 → 1111
    f. 0000 porque −8 + 8 = 0 → 0000

11. No. El valor mayor que puede almacenarse en notación exceso ocho es 7, representado por 1111. Para representar un valor mayor, hay que emplear al menos notación exceso 16 (que utiliza patrones de 5 bits). De forma similar, el valor 6 no se puede representar en notación exceso cuatro (el valor mayor que se puede representar en notación exceso cuatro es 3).

Sección 1.7

1. **a.** $5/8$ **b.** $3\,1/4$ **c.** $9/32$ **d.** $-1\,1/2$ **e.** $-(11/64)$

2. **a.** 01101011 **b.** 01111010 (error de truncamiento)
   c. 01001100 **d.** 11101110 **e.** 11111000 (error de truncamiento)

3. 01001001 ($^{9}/_{16}$) es mayor que 00111101 ($^{13}/_{32}$). He aquí una forma simple de determinar cuál de los dos patrones representa un valor mayor:

   Caso 1. Si los bits de signo son diferentes, el valor mayor será igual que el que tenga el bit de signo igual a 0.

   Caso 2. Si los dos bits de signo son iguales a 0, analizamos la parte restante de los dos patrones de izquierda a derecha, hasta encontrar una posición de bit en la que los dos patrones difieran. El patrón que contenga un 1 en dicha posición representará el valor mayor.

   Caso 3. Si los dos bits de signo son iguales a 1, analizamos la parte restante de los dos patrones de izquierda a derecha, hasta encontrar una posición de bit en la que los dos patrones difieran. El patrón que tenga un 0 en dicha posición representará el valor mayor.

   La simplicidad de este proceso de comparación es una de las razones para representar el exponente en los sistemas de punto flotante mediante notación en exceso, en lugar de mediante notación en complemento a dos.

4. El valor mayor sería $7^{1}/_{2}$, que está representado por el patrón 01111111. En cuanto al valor positivo más pequeño, podríamos decir que existen dos respuestas "correctas". En primer lugar, si nos ajustamos al proceso de codificación descrito en el texto, que requiere que el bit más significativo de la mantisa sea 1 (denominado forma normalizada), la respuesta es $^{1}/_{32}$, que está representado por el patrón 00001000. Sin embargo, la mayoría de las máquinas no imponen esta restricción para valores próximos a 0. En tales máquinas, la respuesta correcta sería $^{1}/_{256}$ que está representado por el valor 00000001.

Sección 1.8

1. Codificación por longitud de recorrido, codificación dependiente de la frecuencia, codificación relativa y codificación por diccionario.
2. 121321112343535
3. Los dibujos animados en color están compuestos por bloques de color homogéneo con bordes bien definidos. Además, el número de colores implicado es limitado.
4. No. Tanto GIF como JPEG son sistemas de compresión con pérdidas, lo que quiere decir que se perderán detalles de la imagen.
5. El estándar de línea base JPEG aprovecha el hecho de que el ojo humano no es tan sensible a los cambios de color como a los cambios de brillo. De este modo, lo que se hace es reducir el número de bits utilizado para representar la información de color, sin una pérdida de calidad apreciable en la imagen.
6. Enmascaramiento temporal y enmascaramiento de frecuencia.
7. Al codificar la información, se realizan aproximaciones. En el caso de datos numéricos, estas aproximaciones se van acumulando al realizar cálculos, lo que puede conducir a la obtención de resultados erróneos. Las aproximaciones no son tan críticas en el caso de las imágenes y del sonido, porque los datos codificados normalmente tan solo se almacenan, transfieren y

reproducen. Sin embargo, si las imágenes o el sonido se reprodujeran, regrabaran y luego se recodificaran de manera repetida, dichas aproximaciones podrían acumularse y terminar produciendo datos no utilizables.

Sección 1.9

1. b, c y e
2. Sí. Si se produce un número par de errores en un byte, la técnica de paridad no permite detectarlos.
3. En este caso, los errores se producen en los bytes a y d de la Cuestión 1. La respuesta a la Cuestión 2 sigue siendo la misma.
4. **a.**

001010111	001101000	101100101
101110010	101100101	000100000
001100001	101110010	101100101
000100000	001111001	101101111
001110101	100111111	

b.

100100010	101001000	101101111
101110111	100111111	100100010
000100000	001000011	001101000
101100101	101110010	001111001
101101100	000100000	001100001
001110011	001101011	101100101
001100100	100101110	

c.

000110010	100101011	100110011
000111101	100110101	100101110

5. **a.** BED **b.** CAB **c.** HEAD
6. Una solución es la siguiente:
   A 0 0 0 0 0
   B 1 1 1 0 0
   C 0 1 1 1 1
   D 1 0 0 1 1

Capítulo 2

Sección 2.1

1. En algunas máquinas, se trata de un proceso en dos pasos consistente en leer primero el contenido de la primera celda del registro y luego escribirlo desde el registro a la celda de destino. En la mayoría de las máquinas, esto se realiza de forma directa en un solo paso, sin utilizar un registro intermedio.
2. El valor que hay que escribir, la dirección de celda en la que hay que escribirlo y el comando necesario para escribir.
3. Los registros de propósito general se utilizan para almacenar los datos inmediatamente aplicables a la operación que se está llevando a cabo; la

memoria principal se emplea para almacenar datos que se necesitarán próximamente y el almacenamiento masivo se utiliza para almacenar datos que probablemente no vayan a ser necesarios en un futuro próximo.

Sección 2.2

1. El término ***mover*** tiene a menudo la connotación de eliminar algo de una ubicación y colocarlo en otra, dejando un hueco detrás. En la mayoría de los casos, en una computadora, dicha eliminación no tiene lugar. En lugar de ello, lo que se suele hacer con el objeto que se está moviendo es copiarlo (o clonarlo) en la nueva ubicación.
2. Una técnica común, denominada direccionamiento relativo, consiste en indicar a qué distancia hay que saltar, en lugar de indicar la dirección absoluta a la que hay que saltar. Por ejemplo, una instrucción podría decirle a la computadora que saltara hacia adelante tres instrucciones o que saltara hacia atrás dos instrucciones. Sin embargo, es preciso darse cuenta de que será necesario modificar dichas instrucciones si posteriormente se insertan instrucciones adicionales entre el origen y el destino del salto.
3. En este caso, podría argumentarse en favor de cualquiera de las dos respuestas. La instrucción se establece en forma de un salto condicional. Sin embargo, puesto que la condición de que 0 sea igual 0 siempre se satisface, el salto se llevará siempre a cabo como si no se hubiera establecido ninguna condición. A menudo podemos encontrarnos con máquinas que tienen este tipo de instrucciones en su repertorio, porque dichas instrucciones proporcionan un diseño eficiente. Por ejemplo, si una máquina está diseñada para ejecutar una instrucción con una estructura tal como "Si ... saltar a...", dicho tipo de instrucción puede emplearse tanto para expresar saltos condicionales como incondicionales.
4. 156C = 0001010101101100
   166D = 0001011001101101
   5056 = 0101000001010110
   306E = 0011000001101110
   C000 = 1100000000000000
5. **a.** Almacenar (STORE) el contenido del registro 6 en la celda de memoria número 8A.
   b. Saltar (JUMP) a la posición DE si el contenido del registro A es igual al del registro 0.
   c. Combinar mediante AND el contenido de los registros 3 y C, almacenando el resultado en el registro 0.
   d. Mover (MOVE) el contenido del registro F al registro 4.
6. La instrucción 15AB requiere que el procesador solicite a la circuitería de memoria el contenido de la celda de memoria situada en la dirección AB. Este valor, después de ser obtenido de la memoria, se almacena en el registro 5. La instrucción 25AB no requiere que se extraiga el valor de la memoria; en lugar de ello el valor AB se almacena directamente en el registro 5.
7. **a.** 2356 **b.** A503 **c.** 80A5

Sección 2.3

1. 34 Hexadecimal
2. **a.** 0F **b.** C3
3. **a.** 00 **b.** 01 **c.** Cuatro veces
4. Se detiene. Es un ejemplo de lo que a menudo se denomina código automodificante. Es decir, el programa se modifica a sí mismo. Observe que las dos primeras instrucciones colocan el valor hexadecimal C0 en la posición de memoria F8, mientras que las dos siguientes instrucciones colocan el valor 00 en la posición F9. Por tanto, en el momento que la máquina alcanza la instrucción contenida en la posición F8, se encuentra con la instrucción de detención (C000) que se ha escrito allí.

Sección 2.4

1. **a.** 00001011 **b.** 10000000 **c.** 00101101
   d. 11101011 **e.** 11101111 **f.** 11111111
   g. 11100000 **h.** 01101111 **i.** 11010010
2. 00111100 con la operación AND.
3. 00111100 con la operación XOR.
4. **a.** El resultado final es 0 si la cadena contiene un número par de 1s. En caso contrario es 1.
   b. El resultado es el valor del bit de paridad, si trabajamos con paridad par.
5. La operación lógica XOR se asemeja a la suma, salvo en el caso de que ambos operandos sean 1, en cuyo caso XOR produce un 0, mientras que la suma es 10. (Por tanto, la operación XOR puede considerarse una operación de suma pero sin acarreo.)
6. Utilice AND con la máscara 11011111 para cambiar minúsculas a mayúsculas. Utilice OR con la máscara 00100000 para cambiar mayúsculas a minúsculas.
7. **a.** 01001101 **b.** 11100001 **c.** 11101111
8. **a.** 57 **b.** B8 **c.** 6F **d.** 6A
9. 5
10. 00110110 en complemento a dos; 01011110 en punto flotante. Lo importante aquí es que el procedimiento utilizado para sumar los valores es diferente dependiendo de la interpretación que le demos a los patrones de bits.
11. Una solución sería la siguiente:

    12A7 (Cargar, LOAD, el registro 2 con el contenido de la celda de memoria A7.)
    2380 (Cargar, LOAD, el registro 3 con el valor 80.)
    7023 (Combinar mediante OR los registros 2 y 3 almacenando el resultado en el registro 0.)
    30A7 (Almacenar, STORE, el contenido del registro 0 en la celda de memoria A7.)
    C000 (Parar, HALT.)

12. Una solución sería la siguiente:

15E0 (Cargar, LOAD, el registro 5 con el contenido de la celda de memoria E0.)
A502 (Rotar, ROTATE, 2 bits hacia la izquierda el contenido del registro 5.)
260F (Cargar, LOAD, el registro 6 con el valor 0F.)
8056 (Combinar mediante AND los registros 5 y 6, almacenando el resultado en el registro 0.)
30E1 (Almacenar, STORE, el contenido del registro 0 en la celda de memoria E1.)
C000 (Parar, HALT.)

Sección 2.5

1. **a.** 37B5
b. Un millón de veces.
c. No. Una página de texto típica contiene menos de 4.000 caracteres. Por tanto, la capacidad de imprimir cinco páginas por minuto indica una velocidad de impresión de no más de 20.000 caracteres por minuto, que es muy inferior a un millón de caracteres por segundo. (Lo importante aquí es que una computadora puede enviar caracteres a una impresora mucho más rápido de lo que la impresora los puede imprimir; por tanto, la impresora necesita una forma de decirle a la computadora que espere.)

2. El disco efectuará 50 revoluciones en un segundo, lo que quiere decir que cada segundo pasarán 800 sectores bajo el cabezal de lectura/escritura. Puesto que cada sector contiene 1.024 bytes, los bits pasarán bajo el cabezal de lectura/escritura a aproximadamente 6,5 Mbps. Por tanto, la comunicación entre la controladora y la unidad de disco tendrá que tener una velocidad al menos igual a esta, para que la controladora no se quede rezagada a la hora de leer datos del disco.

3. Una novela de 300 páginas codificada mediante Unicode ocupa unos 2 MB o 16.000.000 de bits. Por tanto, se requerirán aproximadamente 0,3 segundos para transferir la novela completa a 54 Mbps.

Sección 2.6

1. La cadena de procesamiento contendrá las instrucciones B1B0 (que se está ejecutando), 5002 y quizá también B0AA. Si el valor del registro 1 es igual al valor del registro 0, se ejecutará el salto a la dirección B0 y todo el esfuerzo ya invertido en insertar las instrucciones en la cola de procesamiento se habrá desperdiciado. Por el contrario, lo que no se desperdicia es ningún tiempo, porque ese esfuerzo invertido en cargar dichas instrucciones no ha requerido tiempo adicional.

2. Si no se toma ninguna precaución, la información existente en las posiciones de memoria F8 y F9 se extraerá como si fuera una instrucción, antes de que la parte anterior del programa haya tenido la posibilidad de modificar el contenido de esas celdas.

3. **a.** El procesador que está tratando de sumar 1 a la celda puede leer primero el valor de dicha celda. A continuación, el otro procesador lee el valor de

la celda. (Observe que en este punto ambos procesadores habrán extraído el mismo valor.) Si el primer procesador finaliza ahora la suma y escribe el resultado en la celda antes de que el segundo procesador finalice la resta y escriba su resultado, el valor final contenido en la celda reflejará únicamente la actividad realizada por el segundo procesador.

b. Los procesadores pueden leer los datos de la celda como antes, pero esta vez el segundo procesador podría escribir su resultado antes que el primero. Por tanto, el valor final de la celda solo reflejará la actividad que ha llevado a cabo el primer procesador.

Capítulo 3

Sección 3.1

1. Un ejemplo tradicional sería la cola de personas que están esperando para adquirir las entradas para un determinado evento. En este caso, puede haber alguien que trate de "colarse", lo que violaría la estructura FIFO.
2. Opciones (b), (c) y (e)
3. Los sistemas empotrados se centran a menudo en tareas dedicadas, mientras que los PC son computadoras de propósito general. Los sistemas empotrados suelen tener recursos más limitados que los PC de una antigüedad comparable, pero pueden tener que enfrentarse a límites temporales muy estrictos, con una intervención humana mínima.
4. El tiempo compartido hace referencia a aquella situación en la que hay más de un usuario accediendo a una máquina simultáneamente. La multitarea hace referencia a aquellas situaciones en las que un mismo usuario está realizando más de una tarea al mismo tiempo.

Sección 3.2

1. *Shell:* se comunica con el entorno de la máquina.

   Administrador de archivos: coordina el uso de los dispositivos de almacenamiento masivo de la máquina.

   Controladores de dispositivo: gestionan las comunicaciones con los dispositivos periféricos de la máquina.

   Gestor de memoria: coordina el uso de la memoria principal de la máquina.

   Planificador de tareas: coordina los procesos del sistema.

   Despachador: controla la asignación de tiempo de procesador a los procesos.
2. La línea es vaga y la distinción depende a menudo de quién esté emitiendo el juicio. Hablando en términos aproximados, el software de utilidad realiza tareas básicas y universales, mientras que el software de aplicación lleva a cabo tareas que son específicas de la aplicación a la que la máquina se dedica.
3. La memoria virtual es el espacio imaginario de memoria cuya presencia aparente se crea mediante el procedimiento de intercambiar datos y programas entre la memoria principal y el almacenamiento masivo.

4. Cuando se enciende la máquina, el procesador comienza ejecutando el programa de arranque, que reside en memoria ROM. Este proceso de arranque hace que el procesador transfiera el sistema operativo desde el almacenamiento masivo al área volátil de la memoria principal. Una vez completada esta transferencia, el proceso de arranque hace que el procesador salte al punto de inicio del sistema operativo.

Sección 3.3

1. Un programa es un conjunto de instrucciones. Un proceso es la acción de seguir dichas instrucciones.
2. El procesador completa su ciclo de máquina actual, guarda el estado del proceso actual y configura su contador de programa con un valor predeterminado (que es la posición de la rutina de tratamiento de la interrupción). De este modo, la siguiente instrucción que se ejecute será la primera instrucción de la rutina de tratamiento de interrupciones.
3. Se les podrían dar prioridades más altas, para que el despachador les otorgara un trato preferente. Otra opción sería asignar a los procesos de mayor prioridad unas franjas temporales más largas.
4. Si cada proceso consumiera su franja temporal completa, la máquina podría proporcionar una franja completa a casi 20 procesos en un segundo. Si los procesos no consumen toda su franja temporal, este valor podría ser mucho más alto, pero entonces el tiempo requerido para realizar cada cambio de contexto podría ser más significativo (véase la Cuestión 5).
5. Un total de $5000/5001$ del tiempo de la máquina se invertiría en ejecutar procesos. Sin embargo, cuando un proceso solicita una actividad de E/S, su franja temporal se termina mientras la controladora responde a la solicitud. Por tanto, si cada proceso realizara una de esas solicitudes después de un solo microsegundo de su franja temporal, la eficiencia de la máquina caería a $1/2$. Es decir, la máquina invertiría tanto tiempo realizando cambios de contexto como el que invertiría en ejecutar procesos.

Sección 3.4

1. Este sistema garantiza que el recurso no sea usado por más de un proceso al mismo tiempo. Sin embargo, obliga a que el recurso sea asignado en forma estrictamente alternativa. Una vez que un proceso ha utilizado y liberado un recurso, debe esperar a que otro proceso lo utilice antes de poder acceder de nuevo a él. Esto incluso si el primer proceso necesita el recurso de forma inmediata y el otro proceso no va a necesitarlo durante algún tiempo.
2. Si dos vehículos entran por los extremos opuestos del túnel al mismo tiempo, no serán conscientes de la presencia del otro. El proceso de entrar y encender las señales luminosas es otro ejemplo de región crítica, o en este caso podríamos denominarlo proceso crítico. Con esta terminología, podríamos corregir el fallo diciendo que los vehículos situados en extremos opuestos del túnel podrían llegar a ejecutar el proceso crítico al mismo tiempo.

3. a. Esto garantiza que el recurso no compartible no sea solicitado y asignado de manera parcial; es decir, a cada vehículo se le concede el uso del puente completo o no se le concede.
   b. Esto quiere decir que la asignación del recurso no compartible puede ser cancelada.
   c. Esto hace compartible el recurso no compartible, lo que elimina la competencia.
4. Una secuencia de flechas que forma un bucle cerrado en el grafo dirigido. Es precisamente basándose en esto que se han desarrollado técnicas que permiten a algunos sistemas operativos reconocer la existencia de un interbloqueo y tomar en consecuencia las acciones correctivas oportunas.

Sección 3.5

1. Los nombres y fechas se consideran inadecuados porque son elecciones bastante comunes y no resisten por tanto los intentos de los que se dedican a tratar de adivinar contraseñas. El uso de palabras completas también se considera inadecuado, porque un atacante podría escribir fácilmente un programa para ir probando las palabras contenidas en un diccionario. Además no se recomienda utilizar contraseñas que solo contengan caracteres ya que se construyen a partir de un conjunto de caracteres limitado.
2. Cuatro es el número de patrones de bits distintos que pueden formarse utilizando 2 bits. Si hicieran falta más niveles de privilegio, los diseñadores necesitarían al menos 3 bits para representar los distintos niveles, con lo cual probablemente elegirían usar un total de 8 niveles. De forma similar, la elección natural si queremos menos de 4 niveles de privilegio sería 2, que es el número de patrones que puede representarse con 1 bit.
3. El proceso podría alterar el propio sistema operativo, de modo que el despachador concediera todas las franjas temporales a dicho proceso.

Capítulo 4

Sección 4.1

1. Una red abierta es aquella cuyas especificaciones y protocolos son públicos, lo que permite a los distintos fabricantes producir productos compatibles.
2. Ambos permiten conectar dos buses para formar una red de bus de mayor tamaño. Sin embargo, un puente solo reenvía aquellos mensajes que están destinados al otro lado del puente, mientras que un conmutador tiene múltiples conexiones, cada una de las cuales puede actuar como un puente.
3. Un router es un dispositivo que dirige los mensajes para su transmisión entre las redes que forman una interred.
4. Por ejemplo, una empresa de venta por correo y sus clientes, un cajero automático y los clientes del banco o una farmacia y sus clientes.
5. Hay numerosos protocolos que regulan el flujo de tráfico en las ciudades, la comunicación verbal por vía telefónica y las normas de etiqueta.

6. La computación en cluster implica normalmente la utilización de múltiples computadoras dedicadas, con el fin de proporcionar una computación distribuida de alta disponibilidad o con equilibrado de carga. La computación en retícula tiene un acoplamiento mayor que la computación en cluster y puede implicar el uso de máquinas que se unen a la tarea de computación distribuida solo cuando no están haciendo otra cosa.

Sección 4.2

1. Los ISP de nivel 1 y nivel 2 proporcionan el "núcleo" de las comunicaciones por Internet, mientras que los ISP de acceso proporcionan a sus clientes el acceso a dicho núcleo.
2. El Sistema de nombres de dominio (DNS) es el conjunto de servidores de nombres en Internet que permite traducir las direcciones mnemónicas a direcciones IP (y que permite también realizar la traducción inversa).
3. La expresión 3.6.9 representa el patrón de tres bytes 000000110000 011000001001. El patrón de bits 0001010100011100 se representaría como 21.28 en notación decimal con puntos.
4. Podría haber varias respuestas a esta cuestión. Una sería que ambas notaciones van de lo más específico a lo más general. Las direcciones Internet con el formato mnemónico comienzan con el nombre de una máquina concreta y continúan hasta llegar al nombre del TLD. Las direcciones postales comienzan con el nombre de una persona y van describiendo regiones progresivamente mayores como la ciudad, la provincia y el país. Este orden está invertido en las direcciones IP, que comienzan por el patrón de bits que identifica el dominio.
5. Los servidores de nombres ayudan a traducir direcciones mnemónicas a direcciones IP. Los servidores de correo envían, reciben y almacenan mensajes de correo electrónico. Los servidores FTP proporcionan un servicio de transferencia de archivos.
6. SSH proporciona funciones de cifrado y autenticación.
7. Descargan al servidor inicial de la tarea de enviar mensajes individuales a cada cliente. La técnica P2P traslada esta responsabilidad a los propios clientes (*peers*), mientras que la multidifusión traslada esta tarea a los routers de Internet.
8. Entre los criterios que podríamos considerar se incluyen el coste, la portabilidad, hasta qué punto resulta práctico utilizar nuestra computadora como teléfono, la necesidad de mantener los teléfonos analógicos existentes, la necesidad de emplear servicios telefónicos de emergencia y la fiabilidad y las áreas de servicio de los diversos proveedores implicados.

Sección 4.3

1. Una dirección URL es básicamente la dirección de un documento en la World Wide Web. Un explorador es un programa que ayuda al usuario a acceder a información de hipertexto.
2. Un lenguaje de composición es un sistema para insertar información explicativa en un documento.

3. HTML es un lenguaje de composición concreto. XML es un estándar para el diseño de lenguajes de composición.
4. **a.** <html> indica el principio de un documento HTML.
   b. <head> indica el principio de la cabecera de un documento.
   c. </p> indica el final de un párrafo.
   d. indica el final de un elemento que está vinculado a otro documento.
5. ***Lado de cliente*** y ***lado de servidor*** son términos utilizados para identificar si una actividad se realiza en la computadora del cliente o en la computadora del servidor.

Sección 4.4

1. La capa de enlace recibe el mensaje y se lo entrega a la capa de red. La capa de red determina la dirección en la que hay que reenviar el mensaje y devuelve el mensaje a la capa de enlace para que lo reenvíe. Las capas superiores no son requeridas para el encaminamiento de los mensajes, aunque los routers avanzados pueden emplear las capas de transporte o de aplicación para proporcionar servicios tales como un filtrado selectivo o una calidad de servicio por capas.
2. A diferencia de TCP, UDP es un protocolo no orientado a conexión que no confirma que el mensaje ha sido recibido en el destino.
3. La capa de transporte utiliza los números de puerto del protocolo de transporte para determinar qué unidad dentro de la capa de aplicación debe recibir un mensaje entrante.
4. Realmente no hay nada que lo impida. Un programador en cualquier máquina podría modificar el software que se ejecuta en dicha máquina con el fin de almacenar todos los mensajes que pasen a su través. Esta es la razón por la que los datos confidenciales deben cifrarse.

Sección 4.5

1. El ***phishing*** es una técnica para obtener información confidencial preguntando a los usuarios sus contraseñas, números de tarjeta de crédito, etc. por correo electrónico haciéndose pasar por una entidad legítima, como por ejemplo el banco del usuario o el departamento de informática del campus. Las computadoras no están protegidas frente al ***phishing***; los usuarios deben confiar en su propio sentido común a la hora de decidir si hay que revelar datos confidenciales a otras personas sin la apropiada verificación.
2. La pasarela de una zona es un router que simplemente se limita a reenviar paquetes (partes de mensajes) a medida que estos pasan a su través. Por tanto, un cortafuegos situado en una pasarela no puede filtrar el tráfico según su contenido, sino simplemente según la información de direccionamiento.
3. El uso de contraseñas protege los datos (y por tanto también la información). El uso del cifrado protege a la información.
4. En el caso de un sistema de cifrado de clave pública, el conocer cómo están cifrados los mensajes no permite descifrarlos.

5. Los problemas son internacionales por su propia naturaleza y, por tanto, no están sujetos a las leyes de un único gobierno. Además, las soluciones legales simplemente proporcionan a aquellos que han sufrido algún tipo de daño la posibilidad de recurrir a los tribunales, en lugar de prevenir que esos daños lleguen a producirse.

Capítulo 5

Sección 5.1

1. Un proceso es la actividad de ejecutar un algoritmo. Un programa es una representación de un algoritmo.
2. En el capítulo de introducción hemos hablado de algoritmos para reproducir música, para hacer funcionar a las lavadoras, para construir modelos y para realizar trucos de magia, así como el algoritmo de Euclides. Muchos de los "algoritmos" que llevamos a cabo en nuestra vida cotidiana no son algoritmos porque no se ajustan a nuestra definición formal de algoritmo, como por ejemplo el algoritmo de división de enteros que se ha citado en el texto, o el algoritmo ejecutado por un reloj que continúa haciendo avanzar sus manecillas y dando las horas día tras día.
3. La definición informal falla al no requerir que los pasos estén ordenados y no sean ambiguos. Simplemente enuncia los requisitos de que los pasos sean ejecutables y tengan un determinado fin.
4. Hay que señalar dos aspectos. El primero es que las instrucciones definen un proceso que no termina. En realidad, sin embargo, el proceso terminará por alcanzar un estado en el que ya no habrá más monedas en el bolsillo. De hecho, este podría ser perfectamente el estado inicial. Llegados a este punto, el problema que surge es el de la ambigüedad. El algoritmo, tal como está representado, no nos dice qué es lo que hay que hacer en esta situación.

Sección 5.2

1. Un ejemplo sería la composición de la materia. A un cierto nivel, las moléculas podrían ser las primitivas, pero estas partículas realmente están compuestas por átomos, que a su vez están formados por electrones, protones y neutrones. Además, actualmente sabemos que incluso estas "primitivas" son compuestas.
2. Una vez que un procedimiento está construido correctamente, puede utilizarse como un bloque componente en estructuras de programa más grandes sin tener que considerar de nuevo la composición interna de dicho procedimiento.
3.
```
X ← el valor más grande;
Y ← el valor más pequeño;
while (Y distinto de cero) do
 (Resto ← resto después de dividir X entre Y;
  X ← Y;
  Y ← Resto);
MCD ← X
```

4. Todos los colores de la luz pueden generarse combinando el rojo, el azul y el verde. Por ello, el tubo de rayos catódicos de una televisión está diseñado para generar estos tres colores básicos.

Sección 5.3

1. a.
```
if (n = 1 o n = 2)
  then (la respuesta es la lista que contiene solo el propio valor n)
  else (Dividir n entre 3, obteniendo un cociente q y un resto r.
          if (r = 0)
             then (la respuesta es la lista que contiene q treses)
          if (r = 1)
             then (la respuesta es la lista que contiene (q − 1) treses
                   y dos doses;)
          if (r = 2)
             then (la respuesta es la lista que contiene q treses y
                   un 2)
       )
```
   b. El resultado sería la lista que contiene 667 treses.
   c. Probablemente haya experimentado con valores de entrada pequeños hasta empezar a visualizar un patrón.

2. a. Sí. ***Sugerencia:*** coloque la primera ficha en el centro para evitar el cuadrante que contiene el agujero al mismo tiempo que cubre un cuadrado de cada uno de los otros cuadrantes. Cada cuadrante representa entonces una versión más pequeña del problema original.
   b. El tablero con un solo agujero contiene $2^{2n} - 1$ cuadrados y cada ficha cubre exactamente tres cuadrados.
   c. Los apartados (a) y (b) de esta cuestión proporcionan un excelente ejemplo de cómo conocer la solución de un problema ayuda a resolver otro. Véase la cuarta fase de Polya.

3. Dice, “Esta es la respuesta correcta.”

4. Intentar simplemente ensamblar las piezas sería una técnica abajo-arriba. Sin embargo, examinar la caja del puzzle para ver qué aspecto tiene la imagen añade una componente arriba-abajo a nuestro intento de solución.

Sección 5.4

1. Cambie la prueba en la sentencia `while` por la siguiente “el valor objetivo no es igual a la entrada actual y todavía quedan entradas por considerar”.

2.
```
Z ← 0;
X ← 1;
repeat (Z ← Z + X;
        X ← X + 1)
until (X = 6)
```

3. Esto ha demostrado ser un problema en el lenguaje C. Cuando las palabras clave ***do*** y ***while*** están separadas por varias líneas, los lectores de un programa suelen dudar acerca de la interpretación apropiada de una cláusula ***while***. En particular, el ***while*** situado al final de una sentencia ***do*** suele interpretarse como una sentencia ***while***. Por tanto, la experiencia dice que

es mejor utilizar diferentes palabras clave para representar las estructuras de bucle de precomprobación y postcomprobación.

4. | | | |
|---|---|---|
| Carmen | Alicia | Alicia |
| Genaro | Carmen | Beatriz |
| Alicia | Genaro | Carmen |
| Beatriz | Beatriz | Genaro |

5. Es una pérdida de tiempo insistir en colocar el pivote por encima de una entrada idéntica de la lista. Por ejemplo, haga el cambio propuesto y luego pruebe el nuevo programa con una lista en la que todas las entradas sean iguales.

6.
```
procedure Ordenar (Lista)
N ← 1;
while (N sea menor que la longitud de Lista) do
   (J ← N + 1;
    while (J no sea mayor que la longitud de la Lista) do
        (if (la entrada de la posición J es menor que la entrada
             de la posición N)
             then (intercambiar las dos entradas);
        J ← J + 1)
N ← N + 1)
```

7. La siguiente es una solución ineficiente. ¿Puede hacerla más eficiente?

```
procedure Ordenar (Lista)
N ← la longitud de la Lista;
while (N sea mayor que 1) do
   (J ← la longitud de la Lista;
    while (J sea mayor que 1) do
        (if (la entrada de la posición J es menor que la entrada
             de la posición J − 1)
             then (intercambiar las dos entradas);
    J ← J − 1)
N ← N − 1)
```

Sección 5.5

1. El primer nombre considerado sería Hector, el siguiente sería Lara y el último Jose.
2. 8, 17
3. 1, 2, 3, 3, 2, 1
4. La condición de terminación es “N es mayor o igual que 3” (o “N no es menor que 3”). Esta es la condición para la que no se crea ninguna activación adicional.

Sección 5.6

1. Si la máquina puede ordenar 100 nombres en un segundo, puede realizar ¼ (10.000 − 100) comparaciones en un segundo. Esto significa que cada comparación tarda aproximadamente 0,0004 segundos. En consecuencia, orde-

nar 1000 nombres [lo que requiere una media de ¼ (1.000.000 − 1000) comparaciones] requerirá unos 100 segundos o 1⅔ minutos.

2. La búsqueda binaria pertenece a la clase $\Theta(\lg n)$, la búsqueda secuencial a la clase $\Theta(n)$ y la ordenación por inserción pertenece a la clase $\Theta(n^2)$.
3. La clase $\Theta(\lg n)$ es la más eficiente, seguida de $\Theta(n)$, $\Theta(n^2)$ y $\Theta(n^3)$.
4. No. La respuesta no es correcta aunque pueda parecerlo. Lo cierto es que dos de las tres cartas son iguales por ambos lados. Por tanto, la probabilidad de elegir una de esas cartas es dos tercios.
5. No. Si el dividendo es menor que el divisor, como por ejemplo en 3/7, la respuesta dada es 1, aunque debería ser 0.
6. No. Si el valor de X es cero y el valor de Y es distinto de cero, la respuesta proporcionada no será correcta.
7. Cada vez que se lleva a cabo la prueba de terminación, el enunciado "Suma = 1 + 2 + ... + *K* y *K* menor o igual que *N*" es verdadero. Combinando esto con la condición de terminación "*K* mayor o igual que *N*" se obtiene la conclusión deseada "Suma = 1 + 2 + ... + *N*". Puesto que *K* se inicializa con el valor cero y se incrementa en una unidad cada vez que se pasa por el bucle, su valor terminará por alcanzar el de *N*.
8. Lamentablemente, no. Una serie de problemas están fuera del control del diseño hardware y software, como por ejemplo un mal funcionamiento mecánico o los problemas eléctricos pueden afectar a los cálculos.

Capítulo 6

Sección 6.1

1. Un programa de tercera generación es independiente de la máquina en el sentido de que sus pasos no se enuncian en términos de los atributos de la máquina, como por ejemplo los registros y las direcciones de las celdas de memoria. Por otro lado, es dependiente de la máquina en el sentido de que seguirán produciéndose errores de truncamiento y de desbordamiento aritmético.
2. La principal diferencia es que un ensamblador traduce cada instrucción del programa fuente en una única instrucción del lenguaje máquina, mientras que un compilador suele generar muchas instrucciones en lenguaje máquina para obtener el equivalente a una única sentencia del programa fuente.
3. El paradigma declarativo se basa en desarrollar una descripción del problema que hay que resolver. El paradigma funcional fuerza al programador a describir el problema en términos de las soluciones a otros problemas más pequeños. El paradigma orientado a objetos pone el énfasis en describir los componentes que podemos encontrar en el entorno del problema.
4. Los lenguajes de tercera generación permiten expresar el programa más en términos del entorno del problema y menos en función de las características de la computadora, como hacían los lenguajes de las anteriores generaciones.

Sección 6.2

1. La utilización de una constante descriptiva puede mejorar la accesibilidad del programa.
2. Una sentencia declarativa describe la terminología; una sentencia imperativa describe pasos dentro de un algoritmo.
3. Entero, real, carácter y booleano.
4. Las estructuras `if-then-else` y el bucle `while` son muy comunes.
5. Todos los componentes de una matriz tienen el mismo tipo.

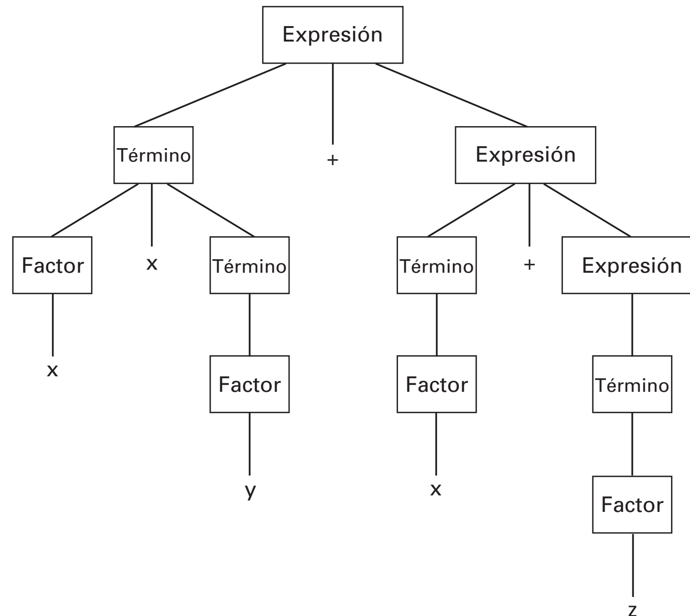
Sección 6.3

1. El ámbito de una variable es el rango del programa en el que la variable es accesible.
2. Una función es un procedimiento que devuelve un valor asociado al nombre de la función.
3. Porque eso es lo que son. Las operaciones de E/S son en realidad llamadas a rutinas que se encuentran dentro del sistema operativo de la máquina.
4. Un parámetro formal es un identificador dentro de un procedimiento. Sirve como contenedor para el valor, el parámetro real, que se pasa al procedimiento en el momento de invocar este.
5. Un procedimiento está diseñado para realizar una acción, mientras que una función está diseñada para generar un valor. Por tanto, el programa es más legible si el nombre de un procedimiento refleja la acción que realiza y el nombre de una función refleja el valor que devuelve.

Sección 6.4

1. *Análisis léxico:* el proceso de identificar los lexemas.
   Análisis sintáctico: el proceso de reconocer la estructura gramatical del programa.
   Generación de código: el proceso de generar las instrucciones que definen el programa objeto.
2. Una tabla de símbolos es el registro de la información que el analizador sintáctico ha obtenido a partir de las sentencias declarativas del programa.
3. En los diagramas sintácticos, los términos que aparecen dentro de los óvalos son terminales. Los términos que requieren una descripción ulterior se incluyen en rectángulos y se denominan “no terminales”.
4. Véase la figura de la página siguiente.
5. Las cadenas que ajustan a la estructura Chachacha están compuestas por una más de las siguientes subcadenas:

   adelante atrás cha cha cha
   atrás adelante cha cha cha
   balanceo derecha cha cha cha
   balanceo izquierda cha cha cha



Sección 6.5

1. Una clase es la descripción de un objeto.
2. Una sería probablemente la clase `ClaseMeteorito` a partir de la cual se construirían los distintos meteoritos. Dentro de la clase `ClaseLaser` podríamos encontrar una variable de instancia denominada `Direccion` que indicaría la dirección hacia la que apunta el láser. Esta variable podría ser utilizada por los métodos `disparar`, `moverDcha` y `moverIzq`.
3. La clase `Empleado` puede contener características relativas al nombre, dirección, antigüedad, etc. del empleado. La clase `EmpleadoTiempoCompleto` podría contener características relativas a planes de pensiones. La clase `EmpleadoTiempoParcial` puede contener características relativas al número de horas de trabajo por semana, el salario por hora, etc.
4. Un constructor es un método especial dentro de una clase que se ejecuta en el momento de crear una instancia de esa clase.
5. Algunos elementos de una clase se designan como privados para evitar que otras unidades de programa obtengan un acceso directo a dichos elementos. Si un elemento es privado, entonces las repercusiones de modificar dicho elemento deberían estar restringidas al interior de la clase.

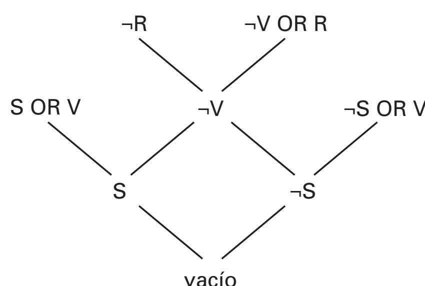
Sección 6.6

1. La lista incluiría técnicas para iniciar la ejecución de procesos concurrentes y técnicas para implementar la comunicación entre procesos.
2. Una sería asignar la responsabilidad a los procesos y otra sería asignar la responsabilidad a los datos. Esta última presenta la ventaja de concentrar la tarea en un único punto del programa.

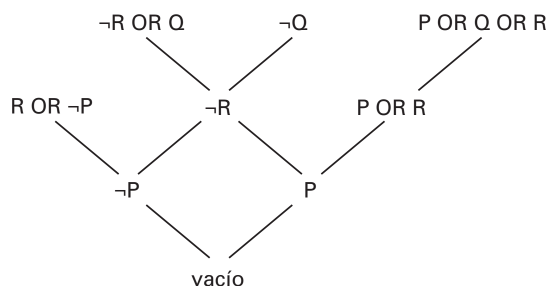
3. Entre ellas se incluyen la predicción meteorológica, el control del tráfico aéreo, la simulación de sistemas complejos (desde reacciones nucleares a control del tráfico de peatones), las redes de computadoras y el mantenimiento de bases de datos.

Sección 6.7

1. R, T y V. Por ejemplo, podemos demostrar que R es una consecuencia añadiendo su negación al conjunto y demostrando que el principio de resolución puede llevarnos al enunciado vacío, como se muestra a continuación:



2. No. El conjunto es incoherente, porque el principio de resolución puede conducir al enunciado vacío, como aquí se muestra:



3.
```
madre(X, Y) :- progenitor(X, Y), hembra(X).
padre(X, Y) :- progenitor(X, Y), varon(X).
```

4. Prolog concluirá que `carolina` es su propia hermana. Para resolver este problema, la regla tiene que incluir el hecho de que `X` no puede ser igual a `Y`, lo que en Prolog se escribe `X \= Y`. Así, una versión mejorada de la regla sería

```
hermano (X, Y) :— X \= Y, progenitor(Z, X), progenitor(Z, Y).
```

lo que quiere decir que `X` es hermano de `Y` si `X` e `Y` no son iguales y tienen un progenitor en común. La siguiente versión, por el contrario, nos dice que `X` e `Y` son hermanos solo si tienen en común ambos progenitores:

```
hermano (X, Y) :— X \= Y, Z \= W
                  progenitor (Z, X), progenitor (Z, Y),
                  progenitor (W, X), progenitor (W, Y).
```

Capítulo 7

Sección 7.1

1. Una secuencia larga de sentencias de asignación no es tan compleja en el contexto del diseño de un programa como unas cuantas sentencias `if` anidadas.
2. ¿Qué tal el número de errores que se encuentren después de un cierto periodo de uso fijo? Un problema con esta solución es que ese valor no puede medirse por adelantado.
3. Lo importante aquí es pensar en cómo pueden medirse las propiedades del software. Una solución para tratar de estimar el número de errores en un segmento de software es colocar intencionadamente algunos errores en el software en el momento de diseñarlo. Luego, después de que el software haya sido supuestamente depurado, comprobamos cuántos de los errores originales conocidos siguen estando presentes. Por ejemplo, si colocamos de forma intencionada siete errores en el software y nos encontramos con que cinco de ellos han sido eliminados después de la depuración, podemos hacer la conjetura de que solo se han eliminado $5/7$ del número total de errores que el software contiene.
4. Entre las posibles respuestas se incluyen el descubrimiento de métricas, el desarrollo de componentes prefabricados, el desarrollo de herramientas CASE o la tendencia hacia la adopción de estándares. Otro, del que trataremos posteriormente en la Sección 7.5, sería el desarrollo de sistemas de notación y modelado como UML.

Sección 7.2

1. Los pequeños esfuerzos realizados durante el desarrollo pueden reportar enormes dividendos durante el mantenimiento.
2. La fase de análisis de requisitos se concentra en lo que el sistema propuesto debe realizar. La fase de diseño se centra en cómo consigue sus objetivos ese sistema. La fase de implementación se concentra en la construcción real del sistema. La fase de pruebas se concentra en asegurarse de que el sistema hace lo que se pretende que haga.
3. Una especificación de requisitos del software es un acuerdo escrito entre un cliente y una empresa de ingeniería de software, en el que se indican los requisitos y las especificaciones del software que hay que desarrollar.

Sección 7.3

1. La técnica tradicional en cascada dicta que las fases de análisis de requisitos, diseño, implementación y de pruebas se realicen de forma lineal. Los modelos más recientes permiten un enfoque más relajado de prueba y error.
2. ¿Qué tal el modelo incremental, el modelo iterativo y XP?
3. El prototipado evolutivo tradicional se realiza dentro de la organización que está desarrollando el software, mientras que el desarrollo de código fuente

abierto no está restringido a una organización. En el caso del desarrollo de código fuente abierto, la persona que controla el desarrollo no determina necesariamente qué mejoras hay que realizar, mientras que en el caso de prototipado evolutivo tradicional es la persona que gestiona el desarrollo del software quien asigna personal a tareas específicas de mejora.

4. Esta es una cuestión que merece la pena considerar con cuidado. Si usted fuera un administrador en una empresa de desarrollo software, ¿le permitirían adoptar la metodología de código fuente abierto para el desarrollo del software que su empresa vaya a comercializar?

Sección 7.4

1. Los capítulos de una novela dependen de los anteriores capítulos, mientras que las secciones de una enciclopedia son en buena medida independientes. Por tanto, una novela tiene un mayor grado de acoplamiento entre sus capítulos que una enciclopedia entre sus secciones. Sin embargo, las secciones de una enciclopedia probablemente tengan un mayor grado de cohesión que los capítulos de una novela.
2. El marcador acumulado sería un ejemplo de acoplamiento de datos. Otros "acoplamientos" que puedan existir incluirían la fatiga, el estado de ánimo, el conocimiento obtenido acerca de la estrategia del oponente y quizá la autoconfianza. En muchos deportes, la cohesión de las unidades se incrementa terminando la acción y reiniciando la siguiente unidad desde el principio. Por ejemplo, en el béisbol cada periodo comienza sin ningún jugador en las bases aún cuando el equipo puede haber finalizado el periodo anterior con ellos. En otros casos, el marcador de cada unidad se controla por separado, como sucede en el tenis donde cada set se pierde o se gana independientemente de lo que haya sucedido en los otros sets.
3. Esta es un cuestión difícil. Desde un cierto punto de vista, podríamos comenzar incluyendo todo en único módulo. Esto daría como resultado muy poca cohesión y ningún acoplamiento. Si entonces empezamos a dividir este único módulo en otros más pequeños, el resultado será un incremento en el grado de acoplamiento. Podemos por tanto concluir que un aumento de la cohesión tiende a incrementar el acoplamiento.

   Por el contrario, suponga que el problema que tenemos entre manos se puede dividir de manera natural en tres módulos de muy alta cohesión, a los que denominaremos A, B y C. Si nuestro diseño original no observara esta división natural (por ejemplo, si agrupáramos la mitad de las tareas de A con la mitad de las tareas de B, etc.), cabría esperar que la cohesión fuera baja y que el acoplamiento fuera alto. En este caso, rediseñar el sistema aislando A, B y C en módulos separados permitiría muy probablemente reducir el acoplamiento entre módulos e incrementar al mismo tiempo la cohesión intermodular.
4. El acoplamiento es el grado de enlazamiento entre módulos. La cohesión es el grado de conexión dentro de un módulo. El ocultamiento de información es la restricción que se impone a la compartición de información.
5. Deberíamos añadir una flecha para indicar que `ControlJuego` debe informar a `ActualizarMarcador` de quién ha ganado y otra flecha en la otra

dirección que indique que `ActualizarMarcador` informará del estado actual (como por ejemplo "fin del set" o "fin del juego") cuando devuelva el control a `ControlJuego`.

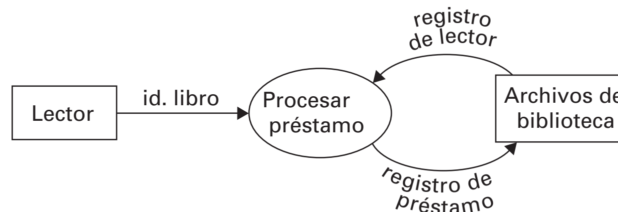
6. Borre todas las flechas horizontales de la Figura 7.5 excepto la primera y la última. Es decir, el juez debe evaluar el servicio del `JugadorA` y enviar directamente el mensaje `ActualizarMarcador` al `Marcador`. (Por supuesto, esto ignora la posibilidad de que exista un segundo servicio. ¿Cómo modificaría el diseño del programa para permitir las faltas dobles?)

7. Un programador tradicional escribe sus programas utilizando sentencias como las que hemos presentado en el Capítulo 6. Un ensamblador de componentes construye los programas enlazando bloques prefabricados denominados componentes.

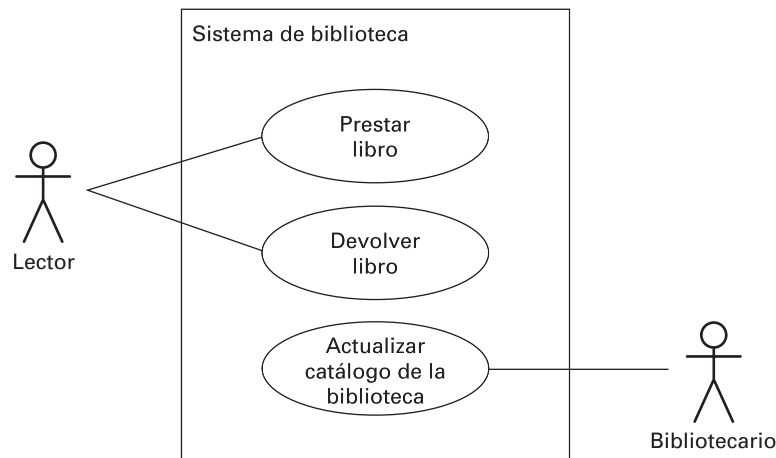
8. Hay muchas respuestas a esta cuestión. Una combinación sería que el calendario fijara automáticamente una alarma en un reloj para notificar al usuario una cita próxima. Además, la aplicación de calendario podría utilizar los componentes de una aplicación de mapas para proporcionar las indicaciones acerca de cómo llegar a la dirección donde tendrá lugar la cita.

Sección 7.5

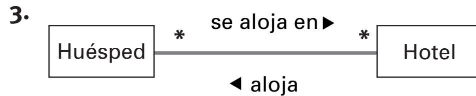
1. Asegúrese de que su diagrama trate con el flujo de datos (no con el movimiento de los libros). El siguiente diagrama indica que las identificaciones de los libros (suministradas por los lectores) y los registros de lectores (extraídos de los archivos de la biblioteca) se combinan para formar los registros de préstamos que se almacenan en los archivos de la biblioteca.



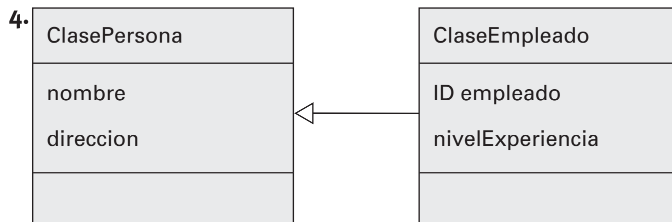
2.



3.



4.



5. Simplemente dibuje un rectángulo alrededor de la figura y añada una etiqueta “sd” en la esquina superior izquierda, como en la Figura 7.13.
6. Los patrones de diseño proporcionan soluciones estandarizadas y bien desarrolladas para la implementación de temas software recurrentes.

Sección 7.6

1. El grupo para el aseguramiento de la calidad del software (SQA) se encarga de controlar e imponer los sistemas de control de calidad adoptados por la organización.
2. Los seres humanos tenemos la tendencia a no anotar los pasos (decisiones, acciones, etc.) que damos durante un proyecto. (También existen problemas relativos a conflictos de personalidades, celos y choques de egos.)
3. Mantenimiento de registros y revisión.
4. El propósito de probar el software es detectar los errores. En un cierto sentido, por tanto, una prueba que no revele ningún error se puede considerar fallida.
5. Una sería considerar el grado de ramificación en los módulos. Por ejemplo, un módulo procedimental que contenga numerosos bucles y sentencias `if-then-else` probablemente será más proclive a errores que un módulo que tenga una estructura lógica simple.
6. El análisis de valores límite sugeriría que probemos el software con una lista de 100 entradas, así como con una lista que no contenga ninguna entrada. También podríamos realizar una prueba con una lista que esté en el orden correcto.

Sección 7.7

1. La documentación toma la forma de documentación de usuario, documentación del sistema y documentación técnica. Puede aparecer en los manuales de acompañamiento, dentro del programa fuente (en forma de comentarios y código bien escrito), en mensajes interactivos que el propio programa escriba en un terminal, en diccionarios de datos y en forma de documentos de diseño, tales como diagramas de estructura, diagramas de clases, diagramas de flujo de datos y diagramas de entidad-relación.

2. Tanto en las fases de desarrollo como de modificación. Lo importante es que las modificaciones deben documentarse tan exhaustivamente como el programa original. (También es cierto que el software se documenta durante la fase de uso. Por ejemplo, un usuario del sistema puede descubrir problemas que, en lugar de ser corregidos, simplemente se comentan en futuras ediciones del manual de usuario del sistema. Además, a menudo se publican libros de carácter práctico después de que el software haya estado utilizándose durante un amplio periodo.)

3. Las diferentes personas tendrán diferentes opiniones sobre esta cuestión. Algunos argumentarán que el programa es el verdadero objetivo de todo el proyecto y que es, por tanto, naturalmente lo más importante. Otros argumentarán que un programa no vale nada si no está documentado, porque si no puede comprenderse, no se podrá ni utilizar ni modificar. Además, con una buena documentación, la tarea de crear el programa puede acometerse de nuevo "fácilmente".

Sección 7.8

1. **a.** ¿Qué tal la capacidad de ajustar la inclinación de una pantalla o la forma de un ratón? En los teléfonos inteligentes, ¿qué tal el uso de pantallas táctiles en lugar de un ratón, o la técnica de inclinar el teléfono con el fin de proporcionar indicaciones de entrada al sistema?
   b. ¿Qué tal la disposición de una ventana en la pantalla, incluyendo el diseño de las barras de herramientas, de las barras de desplazamiento y los menús desplegables? En un teléfono inteligente, ¿acaso no es el inclinar la cámara para apuntar a los elementos de interés algo bastante similar a la forma en la que los seres humanos pensamos?

2. **a.** Sería poco práctico e incómodo utilizar un ratón (o incluso un lápiz óptico) en un teléfono inteligente. Además, el tamaño reducido de la pantalla requiere que los elementos no esenciales de la visualización se restrinjan a un espacio limitado. Por esta razón, las barras de desplazamiento suelen omitirse, y si existen, se muestran en forma de líneas finas.
   b. Un toque deslizante en la pantalla es un gesto natural adaptado a la forma en la que pensamos. Podemos mover papeles u otros elementos deslizándolos sobre una mesa. Algunas personas podrían sostener que esto es bastante más natural que el uso de barras de desplazamiento en una computadora de sobremesa. Aunque las barras de desplazamiento se mueven, en efecto, de la manera esperada, el área que se está desplazando se mueve en la dirección opuesta. Para un usuario que nunca haya usado una computadora, este comportamiento puede parecer anti-intuitivo.

3. Podríamos responder que la diferencia radica en "el papel de las características humanas". Otra buena respuesta sería que el diseño de interfaces se centra en las características externas de un sistema software, más que en las características internas.

4. Las tres que se comentan en el texto son la formación de hábitos, lo estrecho que es nuestro campo de atención y las limitadas capacidades de multi-

procesamiento. ¿Puede imaginarse algunas otras? ¿Qué tal la tendencia a hacer suposiciones?

Sección 7.9

1. El aviso de copyright enuncia quién es el propietario de ese trabajo e identifica al personal autorizado a usar ese trabajo. Todos los trabajos, incluyendo la especificación de requisitos, los documentos de diseño, el código fuente y el producto final, suelen requerir una inversión considerable. Cualquier persona o empresa debe tomar las medidas necesarias para garantizar que se preserven sus derechos de propiedad intelectual y que esa propiedad intelectual no sea utilizada por personas no deseadas.
2. Las leyes de la propiedad intelectual y de patentes benefician a la sociedad porque animan a los creadores de nuevos productos a ponerlos a disposición del público. Sin ese tipo de protección, las empresas dudarían en realizar grandes inversiones en el desarrollo de nuevos productos.
3. Una declaración de renuncia de responsabilidad no protege a una empresa frente a la negligencia.

Capítulo 8

Sección 8.1

1. Lista: un listado de los miembros de un equipo deportivo.
   Pila: la pila de bandejas en una cafetería.
   Cola: la cola de clientes en una cafetería.
   Árbol: el organigrama de muchos gobiernos.
2. Las pilas y colas pueden considerarse como tipos especiales de listas. En el caso de una lista general, las entradas se pueden insertar y eliminar de cualquier posición. En el caso de una pila, las entradas se pueden insertar y eliminar solo por el principio de la pila. En el caso de una cola, las entradas solo pueden insertarse por el final y solo pueden eliminarse por el principio.
3. Las letras de la pila de arriba abajo serían E, D, B y A. Si extrajéramos una letra de la pila, sería la letra E.
4. Las letras de la cola desde el principio al final serían B, C, D y E. Si extrajéramos una letra de la cola, sería la letra B.
5. Los nodos hoja (o terminales) son D y C. B tiene que ser el nodo raíz porque todos los demás nodos tienen padre.

Sección 8.2

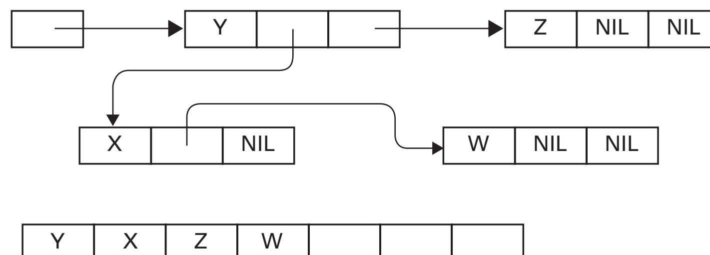
1. Los datos dentro de la memoria principal de una computadora están realmente almacenados en celdas de memoria individualmente direccionables. Las estructuras como matrices, listas y árboles se simulan, para hacer los datos más accesibles a los usuarios de esos datos.
2. Si fuéramos a escribir un programa para jugar a las damas, la estructura de datos que representa el tablero realmente sería una estructura estática por-

que el tamaño del tablero no cambia durante el juego. Sin embargo, si fuéramos a escribir un programa para jugar al dominó, la estructura de datos que representa el patrón de fichas construido sobre la mesa probablemente fuera una estructura dinámica, porque este patrón varía de tamaño y no puede ser predeterminado.

3. Un listín telefónico es básicamente una colección de punteros (números de teléfono) a personas. Las pistas dejadas en la escena de un crimen son punteros (quizá cifrados) que apuntan al perpetrador.

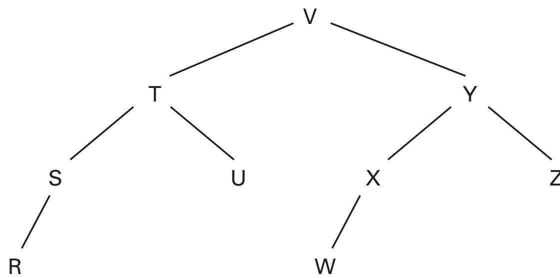
Sección 8.3

1. 5 3 7 4 2 8 1 9 6
2. Si R es el número de filas de la matriz, la fórmula es $R(J - 1) + (I - 1)$.
3. $(c \times i) + j$
4. El puntero inicial contiene el valor NIL.
5.
```
Ultimo ← el último nombre que imprimir
Terminado ← false
Puntero Actual ← el puntero inicial;
while (Puntero Actual not NIL and Terminado = false) do
  (imprimir la entrada a la que apunta Puntero Actual,
   if (nombre que se acaba de imprimir = Ultimo)
     then (Terminado ← true)
   Puntero Actual ← el valor en la celda de puntero
    correspondiente a la entrada a la que apunta
    Puntero Actual)
```
6. El puntero de pila apunta a la celda que se encuentra inmediatamente debajo de la pila.
7. Represente la pila como una matriz unidimensional y el puntero de pila como una variable de tipo entero. A continuación, utilice este puntero de pila para mantener un registro de la posición de la cima de la pila dentro de la matriz, en lugar de la dirección exacta de memoria.
8. Tanto la condición de cola llena como la condición de cola vacía están indicadas por el hecho de que los punteros inicial y final son iguales. Por tanto, hace falta información adicional para distinguir entre esas dos condiciones.
9.

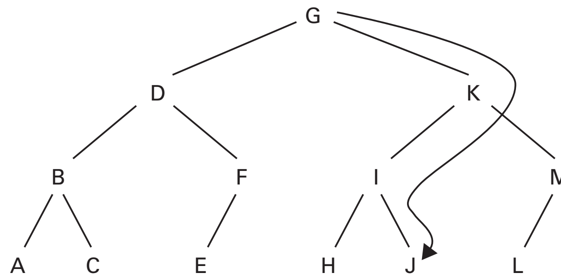


Sección 8.4

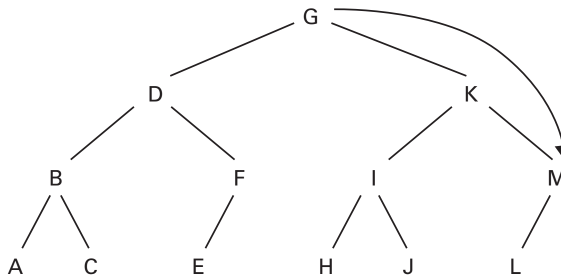
1.



2. Al buscar J:



Al buscar P:



3.

```
procedure  ImprimirArbol(Arbol)

if (puntero raíz de Arbol es no NIL)
   then  (Aplicar el procedimiento ImprimirArbol al
                árbol que aparece como la rama
                izquierda de Arbol;
         Imprimir el nodo raíz de Arbol;
         Aplicar el procedimiento ImprimirArbol al
                árbol que aparece como la rama
                derecha de Arbol.)
```

```
procedure  ImprimirArbol(Arbol)

if (puntero raíz de Arbol es no NIL)
   then  (Aplicar el procedimiento ImprimirArbol al
                árbol que aparece como la rama
                izquierda de Arbol;
         Imprimir el nodo raíz de Arbol;
         Aplicar el procedimiento ImprimirArbol al
                árbol que aparece como la rama
                derecha de Arbol.)
```

Aquí, en el momento
de imprimir K

4. En cada nodo, cada puntero de hijo podría utilizarse para representar una letra distinta del alfabeto. Una palabra podría representarse mediante una ruta descendente por el árbol, junto con la secuencia de punteros que representa la ortografía de la palabra. Un nodo podría marcarse de forma especial si representara el final de una palabra correctamente escrita.

Sección 8.5

1. Un tipo es una plantilla; una instancia de ese tipo es una entidad real construida a partir de esa plantilla. Como analogía, un perro es un tipo de animal, mientras que Lassie y Rex son instancias de dicho tipo.
2. Un tipo de datos definido por el usuario es una descripción de una organización de datos, mientras que un tipo abstracto de datos incluye operaciones para manipular los datos.
3. Un aspecto que hay que resaltar es que podemos elegir entre implementar la lista como una lista contigua o como una lista enlazada. La elección que hagamos afectará a la estructura de los procedimientos que definamos para insertar nuevas entradas, para borrar entradas ya existentes y para encontrar las entradas que busquemos. Sin embargo, esta elección no debe ser visible para el usuario de una instancia de ese tipo abstracto de datos.
4. El tipo abstracto de datos contendrá al menos una descripción de una estructura de datos para almacenar el saldo de la cuenta y procedimientos para efectuar un depósito y para extraer fondos mediante un cheque.

Sección 8.6

1. Tanto los tipos abstractos de datos como las clases son plantillas para construir instancias de un tipo. Sin embargo, las clases son más generales, en el sentido de que están asociadas con el mecanismo de herencia y de que pueden describir también simples conjuntos de procedimientos.
2. Una clase es una plantilla a partir de la cual se construyen objetos.
3. La clase puede contener una cola circular junto con procedimientos para añadir entradas, eliminar entradas, comprobar si la cola está llena y comprobar si la cola está vacía.

Sección 8.7

1. **a.** A5 **b.** A5 **c.** CA
2. D50F, 2EFF, 5FFE
3. 2EA0, 2FB0, 2101, 20B5, D50E, E50F, 5EE1, 5FF1, BF14, B008, C000
4. Al recorrer una lista enlazada en la que cada entrada está compuesta por dos celdas de memoria (una celda de datos seguida de un puntero a la siguiente entrada), una instrucción de la forma DR0S podría utilizarse para extraer el dato, mientras que DR1S podría utilizarse para extraer el puntero a la siguiente entrada. Si se empleara la forma DRTS, entonces podría ajustarse la celda de memoria exacta a la que se está haciendo referencia modificando el valor contenido en el registro T.

Capítulo 9

Sección 9.1

1. El departamento de compras estaría interesado en los registros de inventario para realizar pedidos de materias primas, mientras que el departamento de contabilidad necesitaría la información para llevar las cuentas de la empresa.
2. Un modelo de base de datos proporciona una perspectiva organizativa de una base de datos que es más compatible con las aplicaciones que la organización real. Por tanto, definir un modelo de base de datos es el primer paso para poder utilizar una base de datos como herramienta abstracta.
3. El software de aplicación traduce las solicitudes de los usuarios de la terminología de la aplicación a otra terminología compatible con el modelo de base de datos soportado por el sistema de gestión de base de datos. El DBMS a su vez convierte dichas solicitudes en acciones sobre la base de datos real.

Sección 9.2

1. **a.** Gervasio Salas **b.** Charo Garrido **c.** S26Z
2. Una solución es

```
TEMP ← SELECT from PUESTO
                 where Dept = "PERSONAL"
LIST ← PROJECT TitPuesto from TEMP
```

En algunos sistemas esto da como resultado una lista con títulos de puesto repetidos, dependiendo de cuántas veces aparezca cada puesto dentro del departamento de personal. Es decir, nuestra lista podría contener numerosas apariciones del título Secretaria. Es más común, sin embargo, diseñar la operación PROJECT de modo que se eliminen las tuplas duplicadas de la relación resultante.

3. Una solución es

```
TEMP1 ← JOIN PUESTO and ASIGNACION
                 where PUESTO.IdPuesto = ASIGNACION.IdPuesto
TEMP2 ← SELECT from TEMP1
                 where FechaFin = "*"
TEMP3 ← JOIN EMPLEADO and TEMP2
                 where EMPLEADO.IdEmpl = TEMP2.IdEmpl
RESULT ← PROJECT Nombre, Dept from TEMP3
```

4.
```
select TitPuesto
  from PUESTO
  where Dept = "PERSONAL"
select EMPLEADO.Nombre, PUESTO.Dept
  from PUESTO, ASIGNACION, and EMPLEADO
  where (PUESTO.IdPuesto = ASIGNACION.IdPuesto) and
     ( ASIGNACION.IdEmpl = EMPLEADO.IdEmpl)
      and ( ASIGNACION.FechaFin = "*")
```

5. El propio modelo no proporciona independencia de datos. Esa es una propiedad del sistema de gestión de la base de datos. La independencia de datos se consigue proporcionando al sistema de gestión de los datos la posibilidad de presentar una organización relacional coherente al software de aplicación, aún cuando la organización real de los datos pueda cambiar.
6. Mediante atributos comunes. Por ejemplo, la relación EMPLEADO de esta sección está enlazada con la relación ASIGNACION mediante el atributo `IdEmpl` y la relación ASIGNACION está enlazada con la relación PUESTO mediante el atributo `IdPuesto`. Los atributos como estos utilizados para conectar la relación se denominan en ocasiones atributos de conexión.

Sección 9.3

1. Puede haber métodos para asignar y extraer `FechaInicio`, así como `FechaFin`. También podría proporcionarse otro método para informar del tiempo total de servicio.
2. Un objeto persistente es un objeto que se almacena de manera indefinida.
3. Un enfoque sería establecer un objeto para cada tipo de producto del inventario. Cada uno de estos objetos podría mantener el inventario total de su producto correspondiente, el coste del producto y enlaces con las órdenes pendientes relativas a dicho producto.
4. Como se indica al principio de esta sección, las bases de datos orientadas a objetos parecen manejar los tipos de datos compuestos más fácilmente que las bases de datos relacionales. Además, el hecho de que los objetos puedan contener métodos que adopten papeles activos a la hora de responder cuestiones promete dar a las bases de datos orientadas a objetos una ventaja con respecto a las bases de datos relacionales, cuyas relaciones simplemente almacenan los datos.

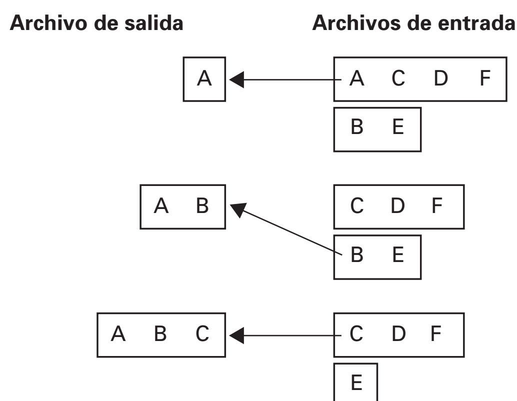
Sección 9.4

1. Una vez que una transacción ha alcanzado su punto de confirmación, el sistema de gestión de la base de datos acepta la responsabilidad de que la transacción completa se realice en la base de datos física. Una transacción que no haya alcanzado su punto de confirmación no dispondrá de tal garantía. Si surgen problemas puede tener que volver a realizarse la transacción.
2. Una solución sería impedir el entremezclado de transacciones durante un instante, para que todas las transacciones actuales puedan completarse. Esto establecería un punto en el que las futuras anulaciones en cascada se detendrían.
3. Se obtendría un saldo de 100 euros si las transacciones se ejecutaran una a continuación de otra. Se obtendría un saldo de 200 euros si se ejecutara la primera transacción después de que la segunda transacción hubiera extraído el saldo original y antes de que almacenara su nuevo saldo. Se obtendría un saldo de 300 euros si la segunda transacción se ejecutara después de que la primera extrajera el saldo original y antes de que la primera almacenara su nuevo saldo.

4. **a.** Si ninguna otra transacción tiene acceso exclusivo, se concederá acceso compartido.
   b. Si otra transacción ya tiene algún tipo de acceso, el sistema de gestión de la base de datos normalmente hará que la nueva transacción espere, o bien podría anular las otras transacciones y proporcionar acceso a la nueva transacción.
5. Se producirá un interbloqueo si dos transacciones obtuvieran acceso exclusivo a diferentes elementos y luego solicitaran acceso al elemento para el que la otra tiene acceso exclusivo.
6. El interbloqueo anterior podría eliminarse anulando una de las transacciones (utilizando el registro) y proporcionando a la otra transacción acceso al elemento de datos que la primera transacción tenía bloqueado.

Sección 9.5

1. Los pasos iniciales serían los siguientes:



2. La idea es dividir en primer lugar el archivo que hay que almacenar en muchos archivos separados, cada uno de los cuales contendrá un registro. A continuación, agrupamos los archivos de un solo registro en parejas y aplicamos el algoritmo de combinación a cada pareja. Esto nos da como resultado la mitad de los archivos, cada uno de ellos con dos registros. Además, cada uno de estos archivos de dos registros estará ordenado. Podemos agruparlos en parejas y aplicar de nuevo el algoritmo de combinación. Con ello, tendremos un conjunto menor de archivos, cada uno de ellos de un tamaño mayor, y además esos archivos estarán ordenados. Continuando de esta forma, terminaremos por obtener un único archivo, que estará formado por todos los registros originales, pero ordenados. (Si en cualquier paso del proceso nos encontráramos con un número impar de archivos, basta con dejar uno de los archivos aparte y emparejarlo con otro de los archivos de mayor tamaño en la siguiente etapa.)
3. Si el archivo está almacenado en cinta o en CD, lo más probable es que su organización física sea secuencial. Sin embargo, si el archivo está almacenado en disco magnético, lo más probable es que esté disperso entre varios sectores del disco y la naturaleza secuencial del archivo será una propiedad

conceptual apoyada en un sistema de punteros o en algún tipo de lista, en la que se anotarán los sectores en los que está almacenado el archivo.

4. En primer lugar, localice el valor clave dentro del índice del archivo. A partir de ahí, obtenga la ubicación del registro objetivo. Después, extraiga el registro almacenado en dicha ubicación.

5. Un algoritmo hash mal elegido provoca un agrupamiento mayor de lo normal y, por tanto, un mayor número de desbordamientos. Dado que el desbordamiento de cada fragmento del almacenamiento masivo está organizado como una lista enlazada, las búsquedas en los registros de desbordamiento son, esencialmente, búsquedas secuenciales.

6. Las asignaciones de fragmentos son las siguientes:

   a. 0 **b.** 0 **c.** 3 **d.** 0 **e.** 3
   f. 3 **g.** 3 **h.** 3 **i.** 3 **j.** 0

   Por tanto, todos los registros hash se asignan a los fragmentos 0 y 3, dejando los fragmentos 1, 2, 4 y 5 vacíos. El problema es que el número de fragmentos que se está utilizando (6) y los valores de clave tienen el 3 como factor común. (Trate de volver a asignar estos valores clave utilizando siete fragmentos y compruebe la mejora que se obtiene.)

7. Lo importante aquí es que estamos aplicando esencialmente un algoritmo hash para dividir a las personas del grupo en 365 categorías distintas. El algoritmo de hash es, por supuesto, el cálculo del cumpleaños de cada uno. Lo sorprendente es que solo hacen falta 23 personas antes de que la probabilidad de que dos de los cumpleaños coincidan sea superior al 50 por ciento. En términos de un archivo hash, esto indica que a la hora de distribuir los registros entre 365 fragmentos disponibles de almacenamiento masivo, el fenómeno del agrupamiento empezará a hacerse notar después de introducir tan solo 23 registros.

Sección 9.6

1. El buscar patrones en un conjunto dinámico de datos es problemático.

2. Descripción de clases: identificación de las características de los subscriptores a una determinada revista.

   Discriminación de clases: identificación de las características que permiten distinguir entre los subscriptores de dos revistas.

   Análisis de agrupamientos: identificación de las revistas que tienden a atraer a subscriptores similares.

   Análisis de asociaciones: identificación de las relaciones entre subscriptores a diversas revistas y entre los diferentes hábitos de compra.

   Análisis de excepciones: identificación de los subscriptores a una revista que no se ajustan al perfil de los subscriptores normales.

   Análisis de patrones secuenciales: identificación de tendencias en la subscripción a revistas.

3. El cubo de datos puede permitir ver los datos de ventas como ventas mensuales, ventas por región geográfica, ventas por clase de producto, etc.

4. Las consultas tradicionales a bases de datos extraen hechos almacenados en la base de datos. La minería de datos busca patrones entre esos hechos.

Sección 9.7

1. Lo importante aquí es comparar su respuesta a esta cuestión con la de la siguiente. Las dos suscitan esencialmente la misma cuestión pero en diferentes contextos.
2. Véase el problema anterior.
3. Podría recibir anuncios publicitarios que de otro modo no habría recibido, pero también podría verse asediado por ofertas indeseadas o convertirse en objetivo de algún grupo de criminales.
4. Lo importante aquí es que una prensa libre puede alertar a los ciudadanos acerca de los abusos o de los abusos potenciales, haciendo así que entre en juego la opinión pública. En la mayoría de los casos citados en el texto, fue la prensa libre la que inició las acciones correctivas al alertar al público acerca de lo que estaba sucediendo.

Capítulo 10

Sección 10.1

1. El procesamiento de imágenes se ocupa del análisis de imágenes bidimensionales, los gráficos 2D tratan con la conversión de formas bidimensionales en imágenes y los gráficos 3D se ocupan de la conversión de escenas tridimensionales en imágenes.
2. La fotografía tradicional produce imágenes de escenas reales, mientras que los gráficos 3D generan imágenes de escenas virtuales.
3. El primero es "construir" la escena virtual. El segundo es capturar la imagen.

Sección 10.2

1. Los pasos son el modelado (construcción de la escena), la reproducción o generación (de una imagen) y la visualización (mostrar la imagen en la pantalla).
2. La ventana de imagen es la parte del plano de proyección que constituye la imagen.
3. Un buffer de imagen es un área de memoria que contiene una versión codificada de una imagen.

Sección 10.3

1. Es un rombo (un cuadrado deformado).
2. Un modelo procedimental es un segmento de programa que controla la construcción de un objeto.
3. La lista podría incluir el terreno cubierto de hierba, un paseo empedrado, los árboles, las nubes, el sol y los actores. Lo importante aquí es darse cuenta de cuál es el ámbito de un grafo de escena (puede contener una gran cantidad de detalles).

4. Representar todos los objetos mediante mallas poligonales proporciona un mecanismo uniforme para el proceso de generación. (En la mayoría de los casos, la generación se implementa como una tarea de generación de parches planos en lugar de generar objetos.)
5. La aplicación de texturas es un medio de asociar una imagen bidimensional con la superficie de un objeto.

Sección 10.4

1. La luz especular es aquella que se refleja "directamente" en una superficie. La luz difusa es luz que se "dispersa" a partir de una superficie. La luz ambiente es aquella que no tiene una fuente definida.
2. El recorte es el proceso de descartar aquellos objetos (y partes de objetos) que no caen dentro del volumen de visualización.
3. Suponga que en mitad de un parche debe aparecer un resalte luminoso. Dicho resalte es provocado por la orientación específica de la superficie en dicho punto del parche. Puesto que el sombreado de Gouraud solo toma en consideración las orientaciones de la superficie a lo largo de las aristas del parche, no será capaz de mostrar ese resalte. Sin embargo, como el sombreado de Phong trata de determinar las orientaciones de la superficie en el interior del parche, sí que podría detectar ese resalte luminoso.
4. La cadena de generación proporciona una técnica estandarizada para generar, lo que permite obtener sistemas de generación más eficientes En particular, la cadena de generación puede implementarse en firmware, lo que quiere decir que el proceso de generación puede realizarse más rápidamente que si implementara dicha tarea mediante software tradicional.
5. El propósito de esta cuestión es hacer pensar al lector sobre las diferencias entre los modelos de iluminación local y global, más que esperar que proporcione una respuesta concreta. Entre las soluciones potenciales que el lector podría proponer se incluyen el colocar copias adecuadamente modificadas de los objetos que hay que reflejar detrás del espejo y considerar el espejo transparente, o tratar de manejar las imágenes que se reflejan en el espejo como si fueran sombras de objetos.

Sección 10.5

1. Solo nos interesan los rayos que terminan por alcanzar la ventana de imagen. Si comenzamos en la fuente luminosa, no sabríamos qué rayos seguir.
2. El trazado de rayos distribuido trata de evitar la apariencia inherentemente brillante generada por el trazado de rayos tradicional. Para ello, lo que hace es trazar múltiples rayos.
3. La radiosidad requiere mucho tiempo de cálculo y no consigue reflejar bien los efectos especulares.
4. Tanto el trazado de rayos como la radiosidad implementan un modelo de iluminación global y ambos hacen un uso muy intenso de la computación. Sin embargo, el trazado de rayos tiende a producir superficies con apa-

riencia brillante, mientras que la radiosidad genera superficies de apariencia apagada.

Sección 10.6

1. Esta pregunta no tiene una respuesta exacta. Si las imágenes perduran durante 200 milisegundos y proyectáramos cinco fotogramas por segundo, cada fotograma se habría desvanecido de nuestros circuitos de percepción en el momento de proyectar el siguiente fotograma. Esto haría, probablemente, que percibiéramos una imagen parpadeante, que resultaría incómoda de contemplar durante un periodo de tiempo prolongado, aunque hay que reconocer que seguiría percibiéndose el efecto de animación. (De hecho, incluso velocidades más bajas permiten generar animaciones burdas.) Observe que la velocidad de cinco fotogramas por segundo está muy por debajo del estándar en el cine, que es de 24 fotogramas por segundo.
2. Un ***storyboard*** es un “esbozo pictórico” de la secuencia de animación deseada.
3. El proceso de interpolación de fotogramas consiste en crear fotogramas que rellenen los huecos entre fotogramas clave.
4. La Dinámica es la rama de la Mecánica que analiza los movimientos que son consecuencia de la aplicación de fuerzas. La Cinemática es la rama de la Mecánica que analiza el movimiento sin tener en cuenta las fuerzas que lo provocan.

Capítulo 11

Sección 11.1

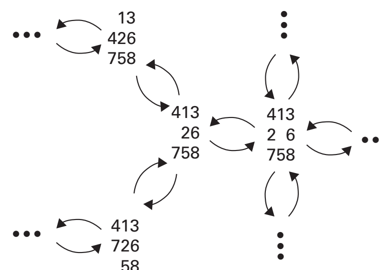
1. Las presentadas en el capítulo incluyen las acciones reflejas, las acciones basadas en el conocimiento del mundo real, las acciones de búsqueda de objetivos, el aprendizaje y la percepción.
2. Nuestro propósito no es dar una respuesta concluyente a esta cuestión, sino usarla para mostrar lo delicado que es realmente el argumento acerca de la existencia de inteligencia.
3. Aunque la mayoría de nosotros diría probablemente que no, sin embargo, si fuera un ser humano el que dispensara los mismos productos en una atmósfera similar probablemente todos diríamos que existe una consciencia, aunque no fuéramos capaces de explicar la diferencia.
4. No existe una respuesta correcta o incorrecta a esta cuestión. La mayoría de los expertos estarían de acuerdo en que la máquina parece al menos ser inteligente.
5. No existe una respuesta correcta o incorrecta a esta cuestión. Es preciso señalar que los bots para mensajería, que son programas diseñados para emular el comportamiento de una persona en un sistema de mensajería, tienen dificultades a la hora de mantener una conversación con sentido incluso durante breves periodos de tiempo. Esos bots para programas de mensajería y salas de chat son fácilmente identificables como máquinas.

Sección 11.2

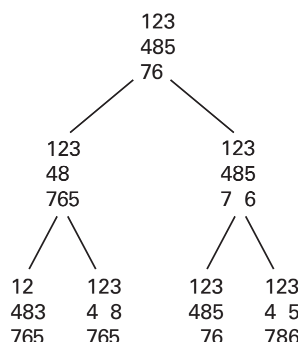
1. En el caso del control remoto, el sistema solo necesita reenviar la imagen, mientras que para usar la imagen con el fin de decidir las maniobras que hay que realizar, el robot debe ser capaz de "comprender" el significado de la imagen.
2. Las posibles interpretaciones para una sección del dibujo no se corresponden con ninguna de las interpretaciones de otra sección. Para incluir este tipo de comprensión dentro un programa, podrían aislarse las interpretaciones permitidas para las diversas uniones de líneas y luego escribir un programa que tratara de encontrar un conjunto de interpretaciones compatibles (una para cada unión). De hecho, si nos paramos a pensar en ello, esto es lo que probablemente nuestros propios sentidos hacen a la hora de tratar de evaluar el dibujo. Probablemente haya detectado usted mismo cómo sus ojos saltaban adelante y atrás entre los dos extremos del dibujo, a medida que sus sentidos intentaban encajar las posibles interpretaciones. (Si le interesa este tema, pruebe a leer acerca de las investigaciones de personas como D. A. Huffman, M. B. Clowes y D. Waltz.)
3. Hay cuatro bloques en la pila, pero solo tres de ellos son visibles. Lo importante en esta cuestión es comprender que este concepto aparentemente simple requiere una cantidad significativa de "inteligencia".
4. ¿Verdad que es interesante? Este tipo de sutiles distinciones en el significado presentan problemas muy complejos en el campo de la comprensión del lenguaje natural.
5. No se sabe si han colocado en la plaza un nuevo banco en el que sentarse o si han abierto una nueva entidad bancaria.
6. El proceso de análisis sintáctico produce estructuras idénticas, pero el análisis semántico reconoce que la cláusula preposicional de la primera frase nos dice dónde se ha construido la valla, mientras que la cláusula de la segunda frase nos dice cuándo se construyó.
7. Son hermanos.

Sección 11.3

1. Los sistemas de producción proporcionan una solución uniforme a un amplio rango de problemas. Es decir, aunque aparentemente sean distintos en su formulación original, todos los problemas reformulados en términos de sistemas de producción se reducen al problema de encontrar una ruta a través de un grafo de estados.
2.



3. El árbol tiene una profundidad de cuatro movimientos. La parte superior tendrá el aspecto siguiente:

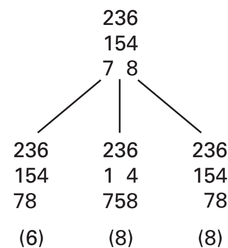


4. La tarea requiere demasiado papel y demasiado tiempo.
5. Nuestro sistema heurístico para la resolución del puzzle de ocho piezas está basado en un análisis de la situación inmediata, al igual que en el caso del escalador. Esta actitud miope es la que permitía a nuestro algoritmo avanzar inicialmente por una ruta incorrecta en el ejemplo de esta sección, al igual que el escalador puede verse en apuros si siempre decide su camino basándose únicamente en el terreno local. (Esta analogía hace que a los sistemas heurísticos basados en información local o inmediata a menudo se les denomine sistemas de escalada de montañas.)
6. El sistema hace girar las piezas 5, 6 y 8 en el sentido de las agujas del reloj o en el sentido contrario a las agujas del reloj hasta alcanzar el estado objetivo.
7. El problema aquí es que nuestro esquema heurístico ignora el valor de mantener el hueco adyacente a las piezas que están descolocadas. Si el hueco está rodeado por piezas que se encuentran en su posición correcta, será necesario mover algunas de esas piezas antes de poder mover las otras que todavía no se encuentran en su lugar adecuado. Por tanto, es erróneo considerar que son correctas todas las piezas que rodean al hueco. Para solventar este problema, podemos primero observar que una pieza que se encuentre en su posición correcta pero bloqueando el hueco a otras piezas en posición incorrecta tendrá que ser desplazada de su posición correcta y luego vuelta a colocar. Así, cada pieza correctamente colocada pero que se encuentre en la ruta entre el hueco y la pieza incorrectamente colocada más próxima aportará al menos dos movimientos a la solución restante. Por tanto, podemos modificar nuestro cálculo del coste estimado del modo siguiente:

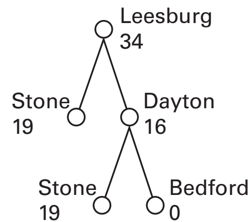
   En primer lugar, calculamos el coste estimado como antes. Sin embargo, si el hueco está totalmente aislado de las piezas incorrectamente colocadas, determinaremos la ruta más corta entre el hueco y una pieza mal colocada, multiplicaremos el número de piezas de esta ruta por dos y sumaremos el valor resultante al coste previamente estimado.

   Con este sistema, los nodos hoja de la Figura 11.11 tendrán un coste estimado de 6, 6 y 4 (de izquierda a derecha), con lo que inicialmente se comenzará a explorar la rama correcta.

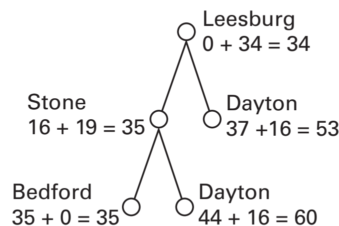
Nuestro nuevo sistema no está libre de errores. Por ejemplo, considere la siguiente configuración. La solución consiste en deslizar la pieza 5 hacia abajo, girar en el sentido de las agujas del reloj las dos filas superiores hasta colocar correctamente esas piezas, mover de nuevo la pieza 5 hacia arriba y finalmente mover la pieza 8 a su posición correcta. Sin embargo, nuestro nuevo sistema heurístico nos diría que comencemos moviendo la pieza 8 porque el estado obtenido por ese movimiento inicial tiene un coste estimado de solo 6, comparado con las otras opciones que tienen un coste de 8.



8. La solución encontrada por el algoritmo de búsqueda de elección del mejor es la ruta que va desde Leesburg a Dayton y luego a Bedford. Esta ruta no es la más corta.



9. La solución encontrada es la ruta que va desde Leesburg a Stone, y luego a Bedford. Esta ruta es la más corta.



Sección 11.4

1. El conocimiento del mundo real es la información acerca del entorno que un ser humano utiliza para comprender y razonar. Desarrollar métodos para representar, almacenar y recuperar esa información es uno de los principales objetivos de la inteligencia artificial.

2. Utiliza la suposición del mundo cerrado.

3. El problema del marco es el problema de actualizar correctamente el almacén de conocimientos de una máquina a medida que se van produciendo

sucesos. La tarea es complicada por el hecho de que muchos sucesos tienen consecuencias indirectas.

4. Imitación, entrenamiento supervisado y refuerzo. El refuerzo no implica una intervención humana directa.
5. Las técnicas tradicionales construyen un único sistema de cómputo. Las técnicas evolutivas implican la construcción de múltiples generaciones de sistemas de prueba, a partir de las cuales puede descubrirse un sistema "correcto".

Sección 11.5

1. Todos los patrones generan una salida igual a 0, excepto el patrón 1, 0, que genera una salida igual a 1.
2. Asigne un peso de 1 a cada entrada y asigne a la neurona un valor de umbral de 1,5.
3. Uno de los problemas principales identificado en el texto es que el proceso de entrenamiento puede oscilar, repitiendo el mismo ajuste una y otra vez.
4. La red derivará hacia la configuración en la que la neurona central está excitada y todas las restantes están inhibidas.

Sección 11.6

1. En lugar de desarrollar un plan de acción completo, el enfoque reactivo consiste en esperar y tomar decisiones a medida que van surgiendo las opciones.
2. Lo importante aquí es pensar en lo amplio que es el campo de la robótica. Abarca todo el campo de la inteligencia artificial, así como numerosos temas de otros campos. El objetivo es desarrollar máquinas verdaderamente autónomas que puedan moverse e interactuar inteligentemente con su entorno.
3. Control interno y estructura física.

Sección 11.7

1. No existe una respuesta correcta o incorrecta.
2. No existe una respuesta correcta o incorrecta.
3. No existe una respuesta correcta o incorrecta.

Capítulo 12

Sección 12.1

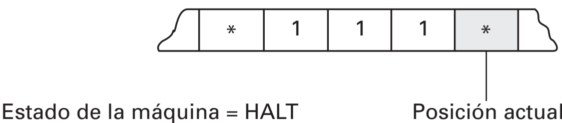
1. Por ejemplo, las operaciones booleanas AND, OR y XOR. De hecho, nosotros hemos utilizado tablas en el Capítulo 1 al presentar estas funciones.
2. El cálculo del pago correspondiente a un préstamo, el área de un círculo o el consumo de gasolina de un automóvil.
3. A tales funciones los matemáticos las denominan funciones transcendentales. Como ejemplos podemos citar las funciones logarítmicas y trigonomé-

tricas. Estos ejemplos concretos pueden calcularse, pero no por medios algebraicos. Por ejemplo, las funciones trigonométricas pueden calcularse dibujando el correspondiente triángulo, midiendo sus lados y solo después recurriendo a la operación algebraica de división.

4. Un ejemplo sería el problema de dividir un ángulo en tres partes iguales. Es decir, no fueron capaces de construir un ángulo que tuviera un tercio del tamaño de otro ángulo dado. Lo importante aquí es que el sistema computacional de los griegos basado en una regla y un compás es otro ejemplo de sistema que presenta limitaciones.

Sección 12.2

1. El resultado es el siguiente diagrama:



2.

Estado actual	Contenido de la celda actual	Valor que hay que escribir	Dirección del movimiento	Estado al que pasar
START	*	*	izquierda	ESTADO 1
ESTADO 1	0	0	izquierda	ESTADO 2
ESTADO 1	1	0	izquierda	ESTADO 2
ESTADO 1	*	0	izquierda	ESTADO 2
ESTADO 2	0	*	derecha	ESTADO 3
ESTADO 2	1	*	derecha	ESTADO 3
ESTADO 2	*	*	derecha	ESTADO 3
ESTADO 3	0	0	derecha	HALT
ESTADO 3	1	0	derecha	HALT

3.

Estado actual	Contenido de la celda actual	Valor que hay que escribir	Dirección del movimiento	Estado al que pasar
START	*	*	izquierda	SUBTRACT
SUBTRACT	0	1	izquierda	BORROW
SUBTRACT	1	0	izquierda	NO BORROW
BORROW	0	1	izquierda	BORROW
BORROW	1	0	izquierda	NO BORROW
BORROW	*	*	derecha	ZERO
NO BORROW	0	0	izquierda	NO BORROW
NO BORROW	1	1	izquierda	NO BORROW
NO BORROW	*	*	derecha	RETURN
ZERO	0	0	derecha	ZERO
ZERO	1	0	derecha	ZERO
ZERO	*	*	sin movimiento	HALT
RETURN	0	0	derecha	RETURN
RETURN	1	1	derecha	RETURN
RETURN	*	*	sin movimiento	HALT

4. Lo importante aquí es que se supone que el concepto de máquina de Turing captura el significado del verbo “computar”. Es decir, en cualquier momento en que se produzca una situación en la que esté teniendo lugar una computación deberían estar presentes los componentes y actividades de una máquina de Turing. Por ejemplo, una persona que esté calculando los impuestos que debe pagar está realizando un cierto grado de computación. La máquina de computación es la persona y la cinta está representada por el papel en el que se anotan los valores.
5. La máquina descrita por la siguiente tabla se detiene si se la inicia con una entrada par, pero nunca se detiene si se inicia con una entrada impar.

Estado actual	Contenido de la celda actual	Valor que hay que escribir	Dirección del movimiento	Estado al que pasar
START	*	*	izquierda	ESTADO 1
ESTADO 1	0	0	derecha	HALT
ESTADO 1	1	1	sin movimiento	ESTADO 1
ESTADO 1	*	*	sin movimiento.	ESTADO 1

Sección 12.3

1.
```
clear AUX;
incr AUX;
while X not 0 do;
  clear X;
  clear AUX;
end;
while AUX not 0 do;
  incr X;
  clear AUX;
end;
```

2.
```
while X not 0 do;
  decr X;
end;
```

3.
```
copy X to AUX;
while AUX not 0 do;
  S1
  clear AUX;
end;
copy X to AUX;
invert AUX; (Véase la Cuestión 1)
while AUX not 0 do;
  S2
  clear AUX;
end;
while X not 0 do;
  clear AUX;
  clear X;
end;
```

4. Si suponemos que X hace referencia a la celda de memoria en la dirección 40 y que cada segmento de programa empieza en la posición 00, tendremos la siguiente tabla de conversión:

	Dirección	Contenido
`clear X;`	00	20
	01	00
	02	30
	03	40

	Dirección	Contenido
`incr X;`	00	11
	01	40
	02	20
	03	01
	04	50
	05	01
	06	30
	07	40

	Dirección	Contenido
`decr X;`	00	20
	01	00
	02	23
	03	00
	04	11
	05	40
	06	22
	07	01
	08	B1
	09	10
	0A	40
	0B	03
	0C	50
	0D	02
	0E	B1
	0F	06
	10	33
	11	40

	Dirección	Contenido
`while X not`	00	20
`0 do;`	01	00
.	02	11
.	03	40
.	04	B1
`end;`	05	WZ
	.	.
	.	.
	.	.
	WX	B0
	WY	00

5. Al igual que sucede en una máquina real se podrían manejar los números negativos mediante un sistema de codificación. Por ejemplo, el bit situado más a la derecha de cada cadena puede utilizarse como signo, empleándose los bits restantes para representar el módulo del valor.
6. La función es una multiplicación por 2.

Sección 12.4

1. Sí. De hecho, este programa se detiene independientemente del valor inicial de sus variables y por tanto debe detenerse si iniciamos sus variables con la representación codificada del programa.
2. El programa solo se detiene si el valor inicial de X termina en 1. Dado que la representación ASCII de un símbolo de punto y coma es 00111011, la versión codificada del programa debe terminar con un 1. Por tanto, el programa es auto-terminante.
3. Lo importante aquí es que la lógica es la misma que en nuestro argumento relativo a que el problema de la detención no tiene una solución algorítmica. Si el pintor pinta su propia casa, entonces no puede hacerlo y viceversa.

Sección 12.5

1. Solo podríamos concluir que el problema tiene una complejidad de orden $\Theta(2^n)$. Si pudiéramos demostrar que el "mejor algoritmo" para resolver el problema pertenece a $\Theta(2^n)$, podríamos concluir que el problema pertenece a $\Theta(2^n)$.
2. No. Como regla general, el algoritmo de orden $\Theta(n^2)$ será mejor que $\Theta(2^n)$, pero para valores de entrada pequeños un algoritmo exponencial suele ser mejor que un algoritmo polinómico. De hecho, es cierto que en ocasiones se prefieren los algoritmos exponenciales a los polinómicos, cuando la aplicación solo utiliza valores de entrada pequeños.
3. Lo importante es que el número de subcomités crece exponencialmente, así que a partir de este punto, la tarea de enumerar todas las posibilidades se hace muy laboriosa.
4. Dentro de la clase de los problemas polinómicos se encuentra el problema de ordenación que puede resolverse mediante algoritmos polinómicos como el de ordenación por inserción.

   Dentro de la clase de problemas no polinómicos tendríamos la tarea de enumerar todos los subcomités que podrían formarse a partir de un determinado comité.

   Cualquier problema polinómico es un problema NP. El problema del viajante es un ejemplo de problema NP que no se ha podido demostrar que sea un problema polinómico.
5. No. Nuestro uso del término ***complejidad*** hace referencia al tiempo requerido para ejecutar un algoritmo, no a lo difícil que pueda ser entender dicho algoritmo.

Sección 12.6

1. $211 \times 313 = 66043$
2. El mensaje 101 es la representación binaria de 5. $5^e = 5^5 = 15625$. 15625 (mod 91) = 64, que es 1000000 en notación binaria. Por tanto, 1000000 es la versión cifrada del mensaje.
3. El mensaje 10 es la representación binaria de 2. $2^d = 2^{29} = 536870912$. 536870912 (mod 91) = 32, que es 100000 en notación binaria. Por tanto, 100000 es la versión descifrada del mensaje.
4. $n = p \times q = 7 \times 19 = 133$. Para determinar d necesitamos un valor entero positivo k tal que $k(p - 1)(q - 1) + 1 = k(6 \times 18) + 1 = 108k + 1$ sea divisible por $e = 5$. Los valores $k = 1$ y $k = 2$ no cumplen esta condición, pero $k = 3$ produce $108k + 1 = 325$, que es divisible por 5. El cociente 65 es el valor de d.

Índice